2022 International Conference on Emerging Techniques in Computational Intelligence (ICETCI) | 978-1-6654-7941-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICETCI55171.2022.9921383

Leveraging Network Similarity Measures for Recommendation Systems

Shagufta Anjum, Mohammed Rayid Ali Masood, Yayati Gupta, and Sanatan Sukhija
Department of Computer Science and Engineering
Mahindra University, École Centrale School of Engineering
Hyderabad, India

***Abstract*—With the growth of e-commerce websites, efficient recommendation systems are desired to reduce the turnaround time for servicing a customer. This study focuses on understanding the various techniques and algorithms that are used to model real-life recommendation systems. We present a recommendation engine for Amazon products that uses collaborative filtering (CF). Given a list of users and their reviews of Amazon products, our CF-based recommendation engine generates a ranked list of the top k products for individual users. The generated recommendations are based on the preferences of similar users and past purchases. We have created two such systems: a memory-based user-item CF system and a model-based CF system that utilizes matrix factorization techniques. Additionally, we also propose a graph-based recommendation technique that generates a list of the most similar products. This method relies on network-based local similarity metrics to generate product suggestions.**

***Index Terms*—recommendation systems, bipartite graph, collaborative filtering, matrix factorization, singular value decomposition, similarity metrics**

I. 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 boosting 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. A recommendation system filters a large amount of data based on its user's historical interests and preferences to predict what items a user would prefer to buy.

In this paper, we attempt to understand these techniques in detail and build our recommendation engine based on both collaborative filtering and graph-based similarity. The rest of the paper is structured as follows: First, we present the relevant literature in this domain, followed by an in-depth review of different types of recommendation systems. Next, we present a description of the dataset and our preprocessing methodology, followed by techniques used to build our recommendation system and the results. Lastly, we describe the challenges we encountered and future scope.

II. Literature Review

The idea of using a bipartite graph in recommendations has been around for a while. Darke et al.[1] made use of link prediction techniques to get a ranked list of the products that are likely to be co-purchased with the queried product. The study compared popular link prediction approaches with several proximity measures to build an efficient recommendation engine. Linden et al.[2] used item-item collaborative filtering for recommending Amazon products. The authors employed cluster models and search-based methods to prune recommendations. Kumar et al.[3] proposed a movie recommendation system called MOVREC. The authors compared their collaborative filtering-based approach against content-based filtering techniques. MOVREC leverages the K-Means clustering algorithm on the movie attributes specified by the user to filter the suggestions. Schafer et al.[4] analyzed several collaborative and content-based filtering techniques. The study emphasized collaborative filtering as one of the major technologies enhancing the adaptive web and highlights the limited effectiveness of content-based recommendation systems. It also presented the idea that it will take decades or more for our technologies to automatically recognize the complexity of what interests people, especially aspects of aesthetic flavour. The study by Patrous et al.[5] focused on finding the best collaborative filtering algorithm on the MovieLens dataset. Additionally, they also explored hybrid recommendation approaches and the associated bottlenecks. A similar study by Makhijani et al.[6] compared several CF techniques for a user-restaurant prediction model. Liu et al.[7] evaluated the rate-prediction performance of Collaborative Filtering for user-based, item-based, and matrix Factorization strategies, against the proposed graph-based Network Inference model.

Our study explores collaborative filtering based techniques for Amazon product purchase ratings. We leverage the ideas from the domain of community detection and link prediction to create our recommendation engine.

III. Background

Recommendation systems aim to predict the interests of users in terms of the products that they are most likely to be interested in. These systems enhance user experience while boosting company sales as well. We can classify these systems

into three broad groups based on the technologies used (see Figure 1). These have been discussed in detail in the upcoming sections.

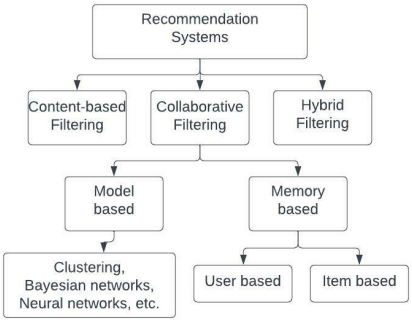

Fig. 1. Types of recommendation systems

A. Content-based methods

A content-based recommender works with data collected from the user, such as rating a product or clicking on a link. Content-based strategies attempt to construct a model based on these available features that specify the discovered user-item interactions. After a model is produced, making new predictions for a user involves studying the user profile and deciding applicable items based on this information to recommend to the user. Content-based strategies suffer far less from the cold start problem (difficulty in making recommendations when the users or the items are new to the dataset and have had very few interactions) than collaborative techniques. In content-based systems, new users or items are defined by their characteristics (content), hence removing the need for interactions between them to make recommendations.

1) User-based approach: To recommend a new product to a user, the user-based approach more or less attempts to find users with the most similar "interactions profile" (nearest neighbours) to indicate items that might be the most popular amongst those neighbours. This approach is considered user-centric since it represents users based on their purchase of items and evaluates distances among users.

To make a recommendation for a given user, each user is first represented by a vector of their interactions with each item. This interaction could be a rating for instance. Then we compute different types of "similarities" between the given user and all other users based on their vectors, such that users with comparable interactions for identical items should be considered as being close to each other in terms of their purchasing habits. Once the similarity values have been computed, we can pick the k-nearest neighbouring users to the given user. The most popular items amongst these neighbors (that haven't been purchased by the user yet) can be recommended to the user.

2) Item-based approach: To make a new recommendation to a user, the concept of the item-based technique is to identify the items that are similar to the ones that the user has positively interacted with in the past, for instance, has given a high rating to. Items are considered similar if the users who have interacted with both those items have interacted with them in the same way. Taking the rating example, this represents the situation where users have rated the two items the same.

This method is considered item-centric because items are represented by the actions that users perform on them, in the form of vectors. The similarity is evaluated like in the user-based approach by calculating the distance between the vectors. To make a recommendation to a given user, we first take the item that this user has rated the highest. We then compute the similarity between this item and all remaining items. Finally, we pick the k-nearest neighbors of this item that the user hasn't purchased before and recommend them.

B. Collaborative Filtering Methods

Collaborative techniques for recommender systems are techniques based entirely on the past interactions among users and items to produce new suggestions. These interactions are represented in the form of a user-item interactions matrix. The primary concept that guidelines collaborative methods is that 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 two sub-classes - memory-based and model-based strategies.

The fundamental benefit of collaborative strategies is that they require no information about users or items. Additionally, the more the users interact with items, the higher the accuracy of new suggestions: new interactions recorded over time provide more information to make the algorithm increasingly effective. However, because collaborative filtering only considers past interactions to make suggestions, it suffers from the "cold start problem": it is difficult to make recommendations to new users or to recommend a new item to users because they have had too few interactions.

1) Memory-based CF: Memory-based strategies directly work with the recorded user-item interactions, assuming no latent model, and are based on nearest neighbours search. For example, we discover the nearest users from a user of interest and recommend the most popular items amongst those neighbours.

2) Model-based CF: Model-based strategies anticipate an underlying "generative" model that 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 using the model have a mathematical meaning that may be difficult to interpret for human beings.

C. Hybrid Methods

A hybrid model attempts to combine functionalities from both content-based and collaborative filtering models to get the best of both worlds. This approach seeks to solve the "cold start" and data sparsity problems. A hybrid approach

can be achieved in multiple 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.

IV. Dataset Analysis and Preprocessing

We used the Amazon Review Data (2018) aggregated by Jianmo Ni from UCSD. 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).

In particular, we used the Electronics reviews dataset, which consists of 20,994,353 reviews. Each row consists of a user-item pairing and all the associated details. Each such pairing is represented as a user-product edge in our graph. The relevant fields to our work are the following:

- reviewerID - ID of the reviewer
- asin - Amazon Standard Identification Number, a unique identifier for each product, e.g. 0000013714
- overall - rating given by the user to the product

After this point, we refer to these fields as User ID (reviewerID), Product ID (asin) and Rating (overall).

A. Network Structure

For the initial understanding of the structure of our data from a network point of view, we represent it as a bipartite graph, with a node for every user and a node for every product. An edge connects users and the products purchased by them. This structure results in what is known as a bipartite graph with all edges between a user node and a product node only. This graph becomes a good starting point to explore the relationship between different pairs of nodes and their proximities to each other.

B. Data Analysis

In our initial dataset, we have 7,824,481 rows, each corresponding to one user review. It consists of 4,201,696 unique users and 476,001 unique products. Due to the huge size of the data, we choose a random subset of 1,000,000 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 the productID vs number of ratings for the top 20 most-rated products as shown in Figure 2. We notice that very few products have a very high number of ratings, while most of them have very few. 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 number of ratings per product is 15.23. To get a full picture, we look at the number of ratings given by each user. The disparity is even higher this time, with a very small fraction of users rating more than one product. The average ratings per user is only 1.15.

We conclude that the data in this form is very sparse; the users and products with few links in the graph will not make

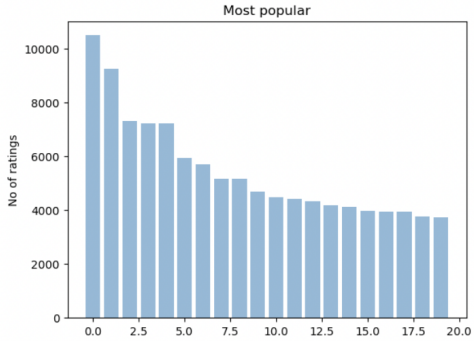

Fig. 2. The distribution of product popularity by the sum of ratings received

a significant contribution to the recommendation system. We attempt to make the data denser by taking a subset of the data created by removing the less significant users and products (the ones with the fewest ratings associated with them). To perform this selection, a minimum threshold needs to be set. We plot the distribution of user ratings (Figure 3) and distribution of product ratings (Figure 4) to give us an idea of the ideal cutoff threshold. Finally, we analyze the ratings. All ratings lie in the

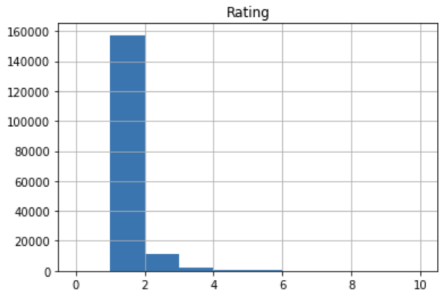

Fig. 3. Distribution of ratings given by users

range from 1 to 5, where 5 is the highest. The mean of the ratings is about 4.01389, indicating that most ratings are on the higher side. A look at the data showed more than half of the ratings to be 5. The distribution of the ratings can be seen in Figure 5.

C. Data Preprocessing

Based on the user and product rating distribution mentioned above, and an analysis of the edges of the bipartite network structure, we decided to remove all products with a rating of less than 10 and all users with a rating of less than 5. In this filtered data, the number of unique users is 12676 and the number of unique products is 26082. This subset is slightly denser than the original data and might be a better choice for making recommendations. The data is randomly divided into a training set (70%) with 69228 reviews and a test set (30%) with 29670 reviews. We execute our prediction algorithms on the training set to predict ratings using both

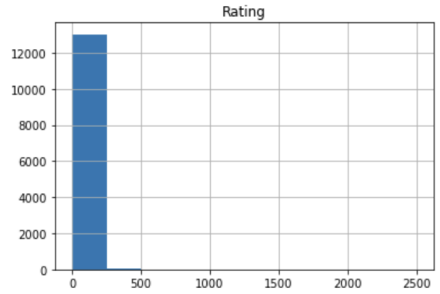

Fig. 4. Distribution of ratings given to items

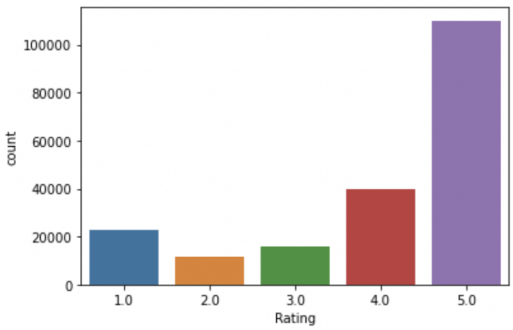

Fig. 5. Frequency of each rating value

memory and model-based collaborative filtering and examine the result using the test set. The details of the approach are outlined in the next section.

V. Approach

Our preference for collaborative filtering (CF) based approach to build our recommendation system is the most broadly used approach to build recommenders and has been successfully employed in various real-life applications. A lot of research has been done on different collaborative filtering models. We implement and compare both variations of this approach - memory-based and model based-techniques. The key feature that distinguishes 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. Similar users 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 reduces when we have sparse data, which is common in most real-world problems, and 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.

A. The Memory-Based Approach

Many similarity measures can be used to calculate the similarity between a pair of a user or product vectors. Some of the most commonly used measurements are cosine similarity, dot product, Euclidean distance, Manhattan distance, and Pearson similarity. In addition to their implementation, we introduce a new way to evaluate similarity using graph-based metrics. In the product unipartite graph

1) Cosine Similarity: Cosine Similarity is a metric that quantifies the similarity among two vectors by the cosine of the angle Θ between the 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.

$$cos(\Theta) = \frac{\mathbf{A}.\mathbf{B}}{\| \mathbf{A} \| \| \mathbf{B} \|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

For our dataset, if ratings are represented by r, each user by u and the set of users by U, then the function for similarity between two items i and j will look like the one below, where $r_{(u,i)}$ denotes the rating given by the user u to item i.

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

To generate the actual predictions, the similarity matrix needs to be combined with the past history of products rated by the users. If I is the set of items and $\overline{r}_i$ is the ratings vector of item i, the prediction score for a user-item pair is given by the following formula.

$$score(u,i) = \frac{\sum_{j}^{I} similarity(i,j)(r_{(u,j)} - \overline{r}_j)}{\sum_{j}^{I} similarity(i,j)} + \overline{r}_i$$

This score represents the predicted rating given by the user to the item.

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 ensure it lies in the required scale of 1-5 (to match the ratings seen on most e-commerce products). 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 users' ratings and then adding 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.

B. The Model-Based Approach

1) Matrix Factorization: Matrix factorization algorithms break down the large, sparse user-item interaction matrix into a product of two smaller and denser matrices: a user-factor matrix (representing users) and a factor-item matrix (representing items). The idea behind the matrix factorization technique is that there is a low dimensional latent space of features where we are can constitute every user and item such that the relationship among them is deduced by calculating the dot product of respective dense vectors in that space. Instead of explicitly feeding those features to our model, the system finds these latent or "hidden" features out by itself and creates its representations of each user and item in terms of these features. These extracted features taken individually have a mathematical meaning but 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 start with a set U of users and a set D of items. Matrix R of size $|U| \times |D|$ is the matrix that includes all of the ratings that been given by users to items. The aim of this method is to extract K latent features from this matrix. Using matrix factorization, the matrix R is broken up into the following matrices:

- Matrix P (of size $|U| \times K$)
- Matrix Q (of size $K \times |D|$)

such that the product of matrices P and Q is $\hat{R}$, which is an approximation to R:

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

In this manner, every row might constitute the power of the institutions among a user and the features. Similarly, every row of Q might constitute the power of the institutions among an item and the features. To obtain the prediction of a rating of an item d_j by u_i, we can to calculate the dot product of the two vectors corresponding to P and Q:

$$\hat{r}_{i,j} = p_i^T q_j = \sum_{k=1}^{K} p_{ik} q_{kj}$$

The next step would be to find the matrices P and Q. This can be done by instantiating the two matrices with a few values and computing how varied their product is from matrix R. Then we try to shrink this difference between them iteratively, using a method known as gradient descent. Gradient descent executes towards optimization, that is, minimization of the difference. This difference between them is called the 'error' between the expected score and the actual score and can be computed using the following equation for each user-item pairing:

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

Note that the error is squared due to the fact the expected rating may be either higher or lower than the actual rating.

To limit the error, we must recognize the direction in which we must adjust the values of p_{i_k} and p_{k_j}. In other words, we must understand the gradient on the current values. Henceforth, we differentiate the above equation with respect to p_{i_k} and p_{k_j} 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 perceived the gradient, we are in a position to now put together the update rules for each p_{i_k} and p_{k_j} as follows:

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

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

α is a constant that decides the rate of descent towards the optimum. Generally, a low value is selected for α, say 0.0002. A bigger value of α may cause issues as we approach the minimum - if we take too big a step, we run into the probability of not accounting for the minimum entirely and instead swaying across it.

A question arises at this point: if we uncover matrices P and Q such that $P \times Q$ approximates R, isn't it that estimations of all of the unseen ratings will be zeros? Furthermore, we aren't simply seeking to come up with P and Q to reproduce R exactly. Instead, we can attempt to limit the errors of the found user-item pairs. To rephrase it, if we allow T to be a set of tuples, each of which is within the form of (u_i, d_j, r_{i_j}), such that T includes all of the perceived user-item pairs altogether with the related scores, we're only intending to limit each e_{i_j} for $(u_i, d_j, r_{i_j}) \in T$ To put it another way, T is our set of training data.

As for the rest of the unknowns, we can conclude their values as soon as the relations among the users, items, and features are learned. Using the above updated regulations, we can iteratively carry out the operation till the error converges to its minimum. We can examine the general error as computed by the usage of the subsequent equation and decide when we have to forestall the process.

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

a) Regularization: The above algorithm is very essential for factorizing a matrix. A common addition to this algorithm is to introduce regularization to avoid overfitting. This is achieved by including a parameter β and altering the squared error:

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

In other words, the new parameter β is used to control the magnitudes of the user-feature and item-feature vectors such

that P and Q would present a very good approximation of R while not having to include massive numbers. In practice, β is set to a few values within the range of 0.02. The new updated regulations for this squared error may be perceived with the help of using a system much like the one defined above. The update rules are redefined as follows:

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

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

b) : Singular Value Decomposition (SVD) Singular Value Decomposition (SVD) is a linear-algebra-based technique that has been normally used as a dimensionality reduction method in machine learning. SVD is a matrix factorization method. It works by reducing 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, SVD is used as a collaborative filtering method. It utilizes 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. It reveals factors of matrices from the factorization of the matrix. It does this by decomposing the matrix into 3 different matrices as given below:

$$A = USV^T$$

Where A of size $m \times n$ is the original matrix, U is a $m \times r$ orthogonal left singular matrix, which represents the connection among users and latent factors, S is a $r \times r$ diagonal matrix, which describes the strength of every latent factor and V is a $n \times r$ diagonal right singular matrix, which suggests the similarity among items and latent factors.

SVD reduces the dimensionality of the utility matrix A by drawing out 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.

C. Graph-based Approach

The third method that we present is quite different from the ones mentioned previously as it uses a pure graph-centric approach to generate recommendations. Here, we focus on the item-item recommendation problem, wherein our program generates a set of products that are most similar to our reference product.

The first step to achieve item-item relationships is to focus only on the product nodes in our previously discussed bipartite graph of users and products. Two products are most obviously related to each other if they have been bought together by the same user. This mutual purchasing of products can be represented as a new graph where all the nodes are products and the edges represent them being purchased together. Such a graph would be a "unipartite" graph as opposed to the previous bipartite graph. To convert the bipartite graph to the unipartite graph, we must first extract all the products purchased by each user. Using this data, we create a list of 'pairings' where each pair would be two products bought together. This pairing can then be easily converted to a graph with the relevant nodes and edges.

The next question to address is how one would measure the similarity between two products in a graph. This brings us to the discussion of graph-based similarity metrics. Several prominent metrics have produced over the years to measure the closeness between a pair of nodes. In this paper, we will use five popular metrics that use local node information to make predictions:

a) Common Neighbors: It is the size intersection of the set of neighbouring nodes of two given nodes u and v. The neighbors of a node are the nodes directly connected to it by an edge. u and v are likely to have an edge between them if they have a lot of common neighbours. This metric is often calculated by taking the square of the adjacency matrix of the graph.

$$CommonNeighbors(u, v) = |u \cap v|$$

b) Jaccard Index: It is the size of the intersection of the neighborhoods of nodes u and v, divided by the size of the union of their neighborhoods. The value of this metric lies between 0 and 1, where the value of 1 means both sets are empty.

$$Jaccard(u, v) = \frac{|u \cap v|}{|u \cup v|}$$

c) Sorenson-Dice Index: It is computed by taking two times the size of the intersection of the neighbours of u and v divided by the sum of the degrees of u and v (the degree of a node is the number of neighbors is has). It is quite similar to the Jaccard Index, but it usually gives higher similarities than its counterpart.

$$Sorenson(u, v) = \frac{2\,|u \cap v|}{|u| + |v|}$$

d) Leicht-Holme-Newman Local Index: It is the size of the intersection of the neighborhoods of u and v, divided by the product of the degrees of u and v. This index is characterized by the higher score it assigns to node pairs that have many common neighbors compared to the expected number of common neighbours.

$$LeichtHolmeNewman(u, v) = \frac{2\,|u \cap v|}{|u| * |v|}$$

e) Preferential Attachment: It is calculated as the product of the degree of the two nodes u and v. The intuition behind this metric is that the more connected a node is, the more likely it is to receive new links. The probability that a new link is connected to a node u is proportional to the degree of u.

$$PrefAttachment(u, v) = |u| * |v|$$

VI. Implementation

A. Collaborative Filtering

We present the relevant data fields using a pivot matrix, where the rows represent users and the columns represent

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 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 they 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.

a) 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 (Figure 6).

```
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
```

Fig. 6. Recommending products for a given user using the model-based approach

B. Graph-based Approach

After the definition and implementation of the graph similarity metrics, we are now ready to apply these to each node pair to generate the similarities between them. Using the list of pairings of products that we had generated earlier, we apply these metrics to each pair of product nodes in the graph and find the similarity scores. We obtain five such score vectors corresponding to each of the five metrics.

Now that our product pair data and the similarity scores for each pair are ready, we can proceed to produce recommendations for any given product. Say we take a product p from our set of products P. For the selected product, we look at its similarity score (for any one of the five metrics) computed with each of the remaining nodes $\in P - \{p\}$. These scores are ranked in decreasing order, and the top k products corresponding to the top k scores are chosen to be recommended since these products have the highest similarity to the chosen product in terms of the particular metric used.

Now our recommendation engine is ready. It takes as input a product and gives as output the list of k recommended products for a chosen metric of similarity (see Figure 7). Notice how many of the recommended products overlap for the different metrics.

```
Recommendations for productId  B0000632GL:

jaccard
['B00005137P', 'B000066TOT', 'B00004U47J', 'B00005T3WF', 'B00006B80L']
common_neighbors
['B00005137P', 'B00005T3WF', 'B00004U47J', 'B00006B80L', 'B00004SB92']
sorenson_dice
['B00005137P', 'B000066TOT', 'B00004U47J', 'B00005T3WF', 'B00006B80L']
leicht_holme_newman_local
['B00006JHZ1', 'B00005137P', 'B000066TOT', 'B000050FH8', 'B00004U47J']
pref_attachment
['B00004SB92', 'B00004VWM3', 'B000056SSM', 'B00004RC2D', 'B00003G1RG']
```

Fig. 7. Output from the graph-based recommendation engine

VII. Evaluation

A. Collaborative Filtering

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.

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

where d_i is the actual rating, $\hat{d}_i$ is the predicted rating and n is the number of ratings. *Results:* Figure 8 shows the RMSE scores: We also compare the average predicted ratings with the average actual ratings for products (Figure 9). The average predicted ratings turned out to be much lower than the average actual ratings.

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

Fig. 8. Comparison of RMSE scores of different approaches.

B. Graph-based Approach

So far, we've generated a unipartite product graph from mutually purchased products, applied similarity metrics to each pair of nodes, and made recommendations using the top k highest similarity scores. The final step in this approach

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

Fig. 9. Comparison of average predicted ratings with average actual ratings

is to quantify the performance of the model. We suggest an evaluation that involves assigning a "performance score" to each node pairing. The computation of this score is detailed below.

For a given pairing of products p and q, we know that they have been purchased together. Now, we generate the recommendations for product p using the top k Jaccard Index values. The performance of the model is evaluated by checking if the mutually purchased product q is included in the recommendations of the product p. Further, the performance score is decided based on the position of q in the list of recommendations:

- If it is in top 5 recommendations, score = 3
- If it is in top 10 recommendations, score = 2
- If it is in top 20 recommendations, score = 1
- Otherwise, score = 0

We can then take the mean of these individual scores to determine the performance for the pair of nodes p and q. After we have computed the scores for all pairs of nodes, we can take the overall mean to determine the performance of the model for the chosen metric. The higher the score, the better the predictions.

VIII. Conclusion

A. Problems faced

1) Collaborative Filtering: 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 scenario 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 item rather than all items they purchase. This bias could skew the results.

2) Graph-based Approach: An obvious shortcoming in this implementation is that the product ratings do not influence the rankings. The products are ranked based only on graph-based proximity, without taking into consideration the ratings of the product. One possible solution to this is to multiply the similarity scores with the average rating of the product being recommended. This would work by 'scaling' the rating proportionate to the average ratings received by the product. This takes into account both the similarity and the rating to determine the final scores, from which we can pick the top k products to recommend.

B. Future Scope

- Comparing the pros and cons of the content and collaborative based approaches, a hybrid approach of combining both these methods may suit a problem better. d model to include a content-based approach along with collaborative filtering so that we can address some of the above-mentioned issues.
- 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 considering additional data will impact the results.

References

[1] Evan Darke, Zhouheng Zhuang, and Ziyue Wang, "Applying Link Prediction to Recommendation Systems for Amazon Products" CS224W Project Final Report.
[2] G. Linden, B. Smith and J. York, "Amazon.com recommendations: item-to-item collaborative filtering," in IEEE Internet Computing, vol. 7, no. 1, pp. 76-80, Jan.-Feb. 2003, doi: 10.1109/MIC.2003.1167344.
[3] Manoj Kumar, D.KYadav, Ankur Singh, Vijay Kr. Gupta," A Movie Recommender System: MOVREC" International Journal of Computer Applications (0975 – 8887) Volume 124 – No.3, August 2015.
[4] Schafer J. B., Frankowski D., Herlocker J., and Sen S., "Collaborative filtering recommender systems," Lecture Notes In Computer Science, vol. 4321, p. 291, 2007.
[5] Z. Salam Patrous and S. Najafi, 'Evaluating Prediction Accuracy for Collaborative Filtering Algorithms in Recommender Systems', Dissertation, 2016.
[6] Rahul Makhijani, Saleh Samaneh, and Megh Mehta , "Collaborative Filtering Recommender Systems " CS229 Project Report.
[7] Xiaoye Liu, "Recommendation System Models in Product Rating Predictions " CS224W Project Report.