Math 3130 Project

Quang Tran

Tiffany Tran

Jeffrey Holloway

27 March 2021

Netflix Recommendation Algorithm

Abstract

As the popularity of streaming services like Netflix or YouTube increases, the demand for more advanced algorithms for recommendation systems parallels. In this report, we will explore the methods used for developing recommendation systems used in Netflix and other similar sites. We will analyze content-based filtering and collaborative filtering systems, as well as discuss MATLAB codes that mimic these processes. We will also explain how certain linear algebra applications are useful for this derivation, including dot product, finding dependencies among rows and columns in matrices, and singular value decomposition.

Introduction

From social media, to streaming services, and even online merchandise (e-commerce), recommendation algorithms offer a tailored experience to the user. Each user gets their own unique experience and recommendations. This increases digital traffic to these sites, which in turn generates a larger profit for the services. But how does a site or streaming service offer such an idiosyncratic experience to each user that visits? Take Netflix for example, a widely popular online streaming service. Once a user logs onto the site, Netflix begins collecting data on the user. They are collecting data on each type of interaction that occurs within the site. They are determining what you are watching and for how long you are watching it. They determine the time of day that you most often watch shows or movies, and even what type of device you are accessing the site from.

To understand how a recommendation algorithm works, one needs to know a basic understanding of linear algebra and how matrices operate. Matrices store data and information in a certain pattern, understanding linear algebra helps us to understand this pattern. Using techniques of linear algebra, one can solve these matrices for specific variables. These variables in turn tell us about the system with which we are studying. In the case of Netflix's recommender algorithm, these variables help to determine which videos are recommended uniquely to each user. But to understand what data fills these matrices, one must look to the recommendation system's hierarchy. There is a hierarchy to this recommender system for data collection that is used by Netflix, Figure 1 below shows this hierarchy.

Commented [TT1]: We should assume that the reader knows about linear algebra as per the project description

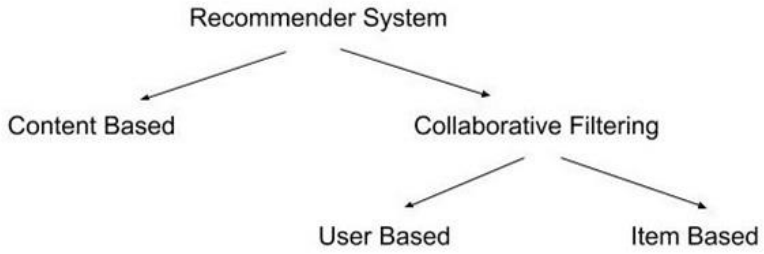

Figure 1: Netflix's Recommender System Hierarchy, source: https://levelup.gitconnected.com/the-mathematics-of-recommendation-systems-e8922a50bdea

Prasad (2020) explains part of this hierarchy in an easily understandable way. First, the content-based recommender system determines what each user would consider as their favorites. This would describe the user's preferences when using Netflix. One of the best ways that Netflix can determine this is when users rate their movies. Netflix would then take these movies and place them in a table with the columns representing the genre or category of movie. In a binary manner, a 1 or 0 would be placed in each cell. This is a numerical representation that shows if the specific movie contains contents from the categories listed in each column. This is also called a one hot encoding approach. An example of this table is listed below in figure 2.

User's ratings

Movie 1	2
Movie 2	10
Movie 3	8

One Hot encoding Approach

	Comedy	Adventure	Super Hero	Sci-Fi
Movie 1	0	1	1	0
Movie 2	1	1	1	1
Movie 3	1	0	1	0

Figure 2: An example of a one hot encoding approach, source: https://levelup.gitconnected.com/the-mathematics-of-recommendation-systems-e8922a50bdea

From this point, we will use a method of linear algebra called a weighted matrix. In the case of the Netflix recommender algorithm, they refer to it as a weighted genre matrix. In linear algebra, a weighted matrix is a method to compare data with differing levels of significance. In the case of this algorithm, the user's rating of the movie gives weight. We multiply this numerical data point by the cells containing the ones or zeros. Netflix would then add the cells in each column together to determine the user's "profile". An example of this is demonstrated in figure 3.

	Comedy	Adventure	Super Hero	Sci-Fi
User	18	12	20	10

Figure 3: An example of a user profile generated by Netflix's recommender algorithm, source: https://levelup.gitconnected.com/the-mathematics-of-recommendation-systems-e8922a50bdea

Finally, the algorithm would then normalize this unique user profile, such as the one referenced above. This is done by adding each element of the row above and then dividing each cell by that sum. From here, we can create two different matrices, one of the "one hot encoded matrix" and the other of the "normalized user profile". We then use methods of linear algebra to multiple the two matrices. A basic understanding of matrix multiplication is as follows. Take the first row of the left matrix and multiply each numerical input by the first column of the right matrix. We then add each product together and continue this process throughout each row and column. A pictorial representation of this can be seen below in figure 3.1.

Commented [TT2]: I think a visual representation of this would be more helpful to understand what is being explained

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} u & v \\ w & x \\ y & z \end{bmatrix}$$

$$= \begin{bmatrix} au + bw + cy & \\ & \end{bmatrix}$$

Figure 3.1: An example of the dot product, source: https://www.intmath.com/matrices-determinants/4-multiplying-matrices.php

The product of these two matrices gives the algorithm what is known as the recommendation matrix. In mathematics, we can use this method, also called the dot product, which utilizes matrix multiplication. In the most basic sense, the dot product tells us how much of one thing is present in another. That is exactly what is occurring here when we take the dot product of the two matrices. Conclusively, the movie with the highest numerical value will be the movie that is recommended to the user. This method does present a problem to Netflix, though. If a user has never seen a specific genre of movie, then there is limited data if at all. This causes problems with the algorithm which in turn lessens the tailored experience. This is a problem that Netflix faces most often with new users. Therefore, there is a different method that is also used by the recommender algorithm, collaborative filtering.

Collaborative filtering, when using the user-based system, determines what is popular by establishing what was enjoyed by your neighbors. When using the item-based approach, the algorithm searches for what is of comparable content of what those in your area enjoyed. The algorithm then tries to recommend these titles to a newer user. In this approach, your geographic data that has been collected when one logs into Netflix is being used. An example of how this item vs. user-based filtering would look is represented below in figure 4. The author, Prasad (2020), distinguishes the solid lines as the user's preference and the dotted lines as what is recommended to the user.

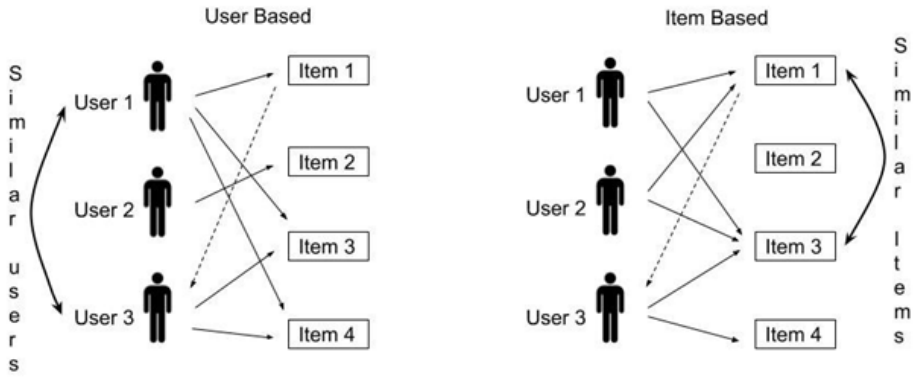

Figure 4: The author's pictorial representation of collaborative filtering, source: https://levelup.gitconnected.com/the-mathematics-of-recommendation-systems-e8922a50bdea

Collaborative filtering is most often used for newer users, often called a "cold start problem". When collaborative filtering is used to determine the similarity between different users, they find past titles that were watched and preferably rated between the users in comparison. From this point, they create a table of users who have watched the same or similar movies, and which lists the users' ranking of that movie in each cell. From here, a weighted matrix is developed in order to rate similar titles. A similarity index is then used to multiply the numerical ratings by. Like before mentioned with the dot product, we then add the numerical entries together to form the recommendation matrix. In a similar manner as above, we then normalize this recommendation matrix by diving its entries by the sum of the similarity index. Exactly as before, the movie with the highest numerical rating is then recommended to the user. This method is far from infallible though, there still are problems that the algorithm will face from new users and a lack of sufficient data.

Mathematical Formulation

Furthermore, we can use the principles mentioned earlier and incorporate them into linear algebra, essentially formulating an algorithm that would characterize the recommendation system in a mathematical way. First, a user rating matrix is developed using the ratings that users have given to movies they have watched. The rows of the matrix represent the users, and the columns represent the movies.

Figure 5

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5

Betty	**1**	**2**	**4**	**1**	**3**
Nancy	**3**	**1**	**1**	**3**	**1**
Deborah	**4**	**3**	**5**	**4**	**4**

In figure 5, these three users have all rated movies 1 through 5. Notice that there is a dependency among rows; Deborah's ratings are the sum of Betty and Nancy's ratings. We can understand this dependency when we think about the problem from a human perspective; say that Betty likes action movies and Nancy likes comedy movies. Then, they would obviously rate the movies based on what their preferences are. We can then predict that Deborah likes both action and comedy movies since her ratings for each movie are a direct sum of Betty's and Nancy's ratings. The same principle can be applied to dependency among columns, but instead of similarities between users, we analyze the similarities between movies (Serrano).

Figure 6

	Movie 1	Movie 2	Twilight
James	**1**	**1**	**1**
Betty	**2**	**4**	**3**
Nancy	**1**	**1**	**1**
Deborah	**3**	**5**	**4**

Notice in figure 6, that the ratings given to the movie "Twilight" are the average of the ratings given to movie 1 and movie 2. This occurs when a movie has content similar to two other movies. For example, if movie 1 is a romance movie, and movie 2 is a vampire movie, then Twilight would be the average of movies 1 and 2 because it is a romance movie about vampires. The dependency among rows and columns makes the notion evidently clear that some humans think alike, and some movies are alike. This evidence is important because it is the basis of collaborative filtering; recommendation systems like those used in Netflix depend on similarities between user data to be able to recommend movies to people the algorithm thinks they will like. This then raises the question: how can we formulate a system that will predict users' ratings on movies, given other users' ratings on the same movie? Otherwise, how can we determine the dependencies among rows and columns? This is where developing an algorithm comes into play (Serrano).

It is important to note that users do not usually rate everything that they watch, and it is also unrealistic to think that users have watched and rated every movie on the platform. It is up to the algorithm to predict what the user rating would be on a movie that has not been rated.

Figure 7

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5
Betty	**3**	**1**	**1**	**3**	**1**
Deborah	**3**	**1**	**1**	**3**	**???**

In figure 7, the missing value for Deborah's rating on Movie 5 can be determined by taking patterns from the previous row and comparing it to the values we already know about Deborah; it's plausible to guess that Betty and Deborah have the same preferences in movies. Therefore, we can say that the missing value is 1, based on Betty's rating of Movie 5. So, we know that finding the dependencies and similarities in a matrix is key to being able to fill in the missing values. The way we can figure out these dependencies is through matrix factorization. To clarify, the goal is to find two smaller matrices that multiply together to give us our initial rating matrix. This is done by characterizing the movies by their genres. We can break down a movie based on its genre and the rating in its genre (with each rating being on a scale from 1 to 5). For example, a movie like "Shrek" can be given a rating of 3 in family movies and a 2 in comedy as shown in figure 8 (the way this rating is generated will be explained later).

Figure 8

	Family Rating	Comedy Rating
Shrek	3	2

Then, we can infer that a user likes family movies, but dislikes comedy movies based on their past ratings of other movies, as shown in figure 9 (this guess is from the same process that will be explained later).

A user liking a movie is represented as a "1", while a user disliking a movie is represented with a "0". Let us say we don't know what the user's rating is for the movie Shrek. We can predict the user's rating by taking the dot product of the two row vectors:

	Family Preference	Comedy Preference
Example User	1	0

Figure 9

$$(1\ x\ 3)\ +\ (0\ x\ 2)\ =\ 3$$

Notice that the 3 is carried over, while the 2 is zeroed out. This makes sense, as this user likes family movies but dislikes comedy movies, so the weight of the family movie rating will be carried over while the comedy movie rating will not be considered at all. Therefore, our inferred rating of Shrek will be 3. Let us look at this approach on a bigger scale, given more examples shown in figure 10 and 11:

Figure 10

	Family Preference	Comedy Preference
Harry	1	0
Ted	0	1
Mike	1	0
Sarah	1	1

Figure 11

	Movie 1	Movie 1	Movie 3	Movie 4	Movie 5
Family Rating	3	2	4	1	1
Comedy Rating	1	3	1	2	2

By multiplying these two matrices together (figure 10, a 4 x 2 matrix, multiplied by figure 11, a 2 x 5 matrix) we get the resulting 4 x 5 matrix:

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5
Harry	3	2	4	1	1
Ted	1	3	1	2	2
Mike	3	2	4	1	1
Sarah	4	5	5	3	3

Notice that each rating value is calculated by matrix multiplication of two matrices that describe the genre preferences of each user, and the rating of the specific genre of each movie. This is how matrix factorization happens. We express the ultimate, user rating matrices into two smaller factors, one matrix with user preferences of each genre, and another matrix with each movie with their genre ratings. Also, notice Harry and Mike have the same preferences, both liking family movies and disliking comedy movies. Their same preferences are expressed in the user rating matrix as well, as seen in figure 12, where the ratings in every single movie for Harry and Mike are the same.

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5
Harry	3	2	4	1	1
Mike	3	2	4	1	1

Figure 12

Figure 13

Also notice how movie 4 and movie 5 in figure 11 have the same ratings for family and comedy, a 1 and a 2, respectively. This is also expressed in the user rating matrix, as movie 4 and movie 5 have the same ratings in figure 13.

	Movie 4	**Movie 5**
Harry	1	1
Ted	2	2
Mike	1	1
Sarah	2	2

It is clear that our factor matrices express all the patterns and data that our original user rating matrix would have. Factorizing our given user rating matrix is useful for several reasons. One, because we can express a huge amount of data in just two smaller factor matrices (Serrano). This saves a huge amount of data if we consider it on a scale as big as services such as Netflix, Hulu, Amazon, etc. Millions of numbers can be expressed in potentially only a couple thousand numbers. Two, when we factorize our user rating matrices, we can also guess numbers for ratings that we don't know yet.

Taking a step back, we should analyze the biggest question right now. How did we find this factorization? How did we predict user's likes and dislikes? How did we calculate the genre ratings for each movie? We do this through machine learning. By developing an algorithm that predicts the factor matrices until their product comes "close enough" to our original user rating matrix, we are creating a "guess and check" method that will give us what we need to predict the unknown ratings for users. The first step of this process is by using the values that are already given to us and use them to our advantage, such as user ratings for movies, dependencies among rows, and dependencies among columns. Then, we develop an algorithm that will predict the 2 smaller matrices (user preference matrix, movie genre rating matrix) that multiply together to create the ultimate user rating matrix. The machine keeps guessing until it reaches a value (product of the 2 guessed matrices) that is "close enough" to the user rating matrix that was already given to us (Serrano). Once we have the 2 smaller matrices, we can predict the missing user ratings that were not given to us in the beginning. To put it into simpler terms, users will rate a movie they watch. These ratings get put into one big matrix, rows are the users, and the columns are each movie. It is up to the machine to find the underlying patterns in what seems like a matrix with random numbers, but essentially, they are key to solving the recommendation algorithm problem. These patterns are because people rate things based on their preferences, the similarity of movies, etc. The patterns are how we go from this matrix, something that is given to us:

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5
Harry	3	2	???	1	1
Ted	1	3	1	???	2
Mike	3	???	4	1	1
Sarah	???	5	5	???	2

	Family Preference	Comedy Preference

Harry	1	0
Ted	0	1
Mike	1	0
Sarah	1	1

which is then factorized into two separate matrices:

	Movie 1	Movie 1	Movie 3	Movie 4	Movie 5
Family Rating	3	2	4	1	1
Comedy Rating	1	3	1	2	2

and then taking the product of these two matrices to find the completed, user rating matrix.

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5
Harry	3	2	4	1	1
Ted	1	3	1	2	2
Mike	3	2	4	1	1
Sarah	4	5	5	2	2

Realistically speaking, we would not use 1's and 0's to determine someone's preference in a genre, and the ratings would not be perfect integers. These numbers would more than likely be float numbers with intricate values. The user rating matrix for sites like Netflix would also be significantly larger than displayed above, where the streaming service boasts around 204 million subscribers (Iqbal) and thousands of movies. This results in an astronomically large user rating matrix, and therefore a lot of issues with storage and memory. Matrix factorization is also another direct solution to this problem; in only having to store 2 matrices that represents data for over a hundred million users, there is a lot of space saved and it makes it significantly easier to retrieve data.

In broader terms, this algorithm does not have to be only applied to Netflix. This can be applied to any other e-commerce, streaming, or social media site. Instead of ratings, the beginning user rating matrix can be developed by implicit feedback (Cates).

Implicit feedback is determined by the number of times someone clicks on something, how many times they have played a song, or the amount of time someone has watched a video. Explicit feedback is given by the direct ratings, likes, favorites that are given to an item, show, video, etc. Then, this

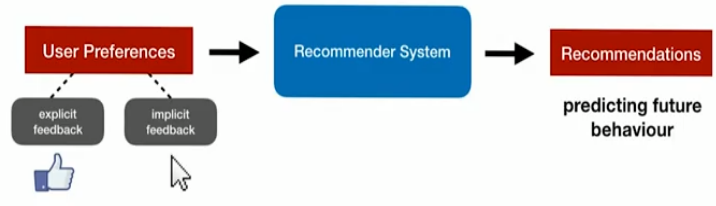

Figure 14: Chart flow diagram of how content is recommended. Source: https://www.youtube.com/watch?v=v_mONWiFv0k

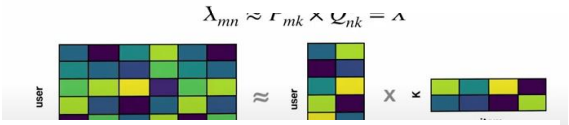

Figure 15: Visual representation of the user-factor and item-factor matrices. Source: https://www.youtube.com/watch?v=v_mONWiFv0k

information is fed to a recommendation system, which factorizes the information into two matrices. The machine does not know what the 2 smaller matrices represent. But the user rating matrix, which is given to us, has underlying patterns that would essentially create the two smaller matrices. These patterns create matrices that represent things such as user's likes/dislikes, or the genre ratings given to movies, but the machine does not know that other than the fact that one factor characterizes patterns in users and one factor characterizes patterns in the items, movies, etc. These matrices are regarded as the user-factor matrix and the item-factor matrix, respectively. Through the data gathered by implicit and explicit feedback, we can then think of specific algorithms that would start to form our item-factor and user-factor matrices (Cates).

We can also use processes like singular value decomposition in collaborative filtering systems, which is a method that "has been generally used as a dimensionality reduction technique in machine learning" (Kumar). Moreover, with the problems of having a dataset of over a hundred million users, we want to be able to use singular value decomposition to be able to "shrink" its dimension "from N-dimension to K-dimension (where K < N)" (Kumar). Singular value decomposition (SVD) is the decomposition of an $m\ x\ n$ matrix A into three matrices, $A = USV^T$, where U is an orthogonal $m\ x\ m$ matrix that represents "the relationship between users and latent factors" (Kumar), V is an also orthogonal, $n\ x\ n$ matrix that represents "the similarity between items and latent factors" (Kumar), and S is an $m\ x\ n$ diagonal matrix with singular, positive values that represents the weight of each latent factor, ranked from greatest to least value from left to right (Vozalis and Margaritis). Essentially, these latent factors are our underlying user preferences and movie similarities in the dataset. From then, we can express U, S, and V as matrices with dimensions $m\ x\ r, r\ x\ r, and\ r\ x\ n$ respectively, as we only want to focus on the first r diagonal entries of S where $s_1 \geq s_2 \geq \cdots \geq s_r$ and $s_i > 0$. SVD "provides the best low-rank approximation of the original matrix A. By retaining the first $k \ll r$ singular values of S and discarding the rest, which can be translated as keeping the $k\ largest$ singular values… we reduce the dimensionality of the data representation" (Vozalis and Margaritis). We then state the resulting S matrix as S_k, while U and V are then derived by removing $r - k$ columns from U and $r - k$ rows from V, where A_k can be represented with K-dimensionality (Vozalis and Margaritis):

$$A_k = U_k\ x\ S_k\ x\ (V_k)^T$$

Examples and Numerical Results

Content-Based Filtering:

In our MATLAB code for content-based filtering, we started off with building a preference matrix with 0's and 1's. The matrix shown in figure 17 has individual users for rows and various movie genres for columns. Moving forward, the variables are defined sequentially, with column 1 corresponding to genre 1, column 2 with genre 2, and so on. The same is true with rows, with row n corresponding to user n. The variables can be anything related to users and items, such as row shoppers by column brands.

```
listOfGenres =

  1×2 string array

    "Family"    "Comedy"
```

Figure 16

```
userPreferenceMatrix =

     1     0
     0     1
     1     0
     1     1
```

Figure 17

Next up, there would be a matrix with a "movie score" for each genre of movies shown. The information stored in the matrix is read row genre score by columns movies. For example, in this case, in column two of figure 18, the vector shown below is <3,1>. This tells us that movie 2, has a genre score of 3 for genre 1, and a genre score of 1 for genre 2. With genre 1 being "Family" and genre 2 being "Comedy", as shown above, it can be concluded that movie 2 is more suited to be a Family movie rather than a Comedy movie. Therefore, from the user preference matrix, this movie would suit someone who likes family movies (column 1) but dislikes comedy movies (column 2). From the user preference matrix, this would be user 1 and user 3, as shown in figure 17.

By applying matrix multiplication on figure 17 and 18, the resulting recommendation matrix is formed below in figure 19. The rows of the matrix represent the individual users, users 1 through 4, and the columns represent the individual movies, movies 1 through 5, this time, with an overall preference rating of the movie itself. Each element is based on dot product between the user preference vector, and the movie genre rating vector. For example, in the recommendation matrix, element (4,3) is 7. This is formulated from the dot product between the row vector 4 of the user preference matrix, and column vector 3 of the movie genre score matrix. The resulting vectors are $< 1 \quad 1 >$ and $< 4 \quad 3 >^{T}$ The dot product results in $(1x\ 4) + (1\ x\ 3) = 7$.

```
randMovieGenreMatrix =

     2     3     4     4     2
     2     1     3     2     1
```

Figure 18

Commented [TT3]: use LaTeX it would be more coherent with the rest of the paper

```
reccomendationMatrix =

     2     3     4     4     2
     2     1     3     2     1
     2     3     4     4     2
     4     4     7     6     3
```

Figure 19

In general, for content-based filtering, the downside is that it requires preferences data from the user to be collected directly. In other words, the application requires its users to tell it what their personal preferences are. Especially in this case, as this is a "yes or no" preference, like or dislike, with no middle ground. Because of that, the resulting matrix may not be perfectly accurate to everyone's likings.

Collaborative Filtering:

In contrast to content-based filtering, where two submatrices are multiplied together to build the recommendation system matrix, collaborative filtering works backwards. Collaborative filtering

starts off with the recommendation matrix, typically with missing or unideal elements, and factorizes the matrix into two smaller submatrices. Just like in content-based filtering, these two submatrices contain information about the users, their preferences, as well as correlation to each individual movie.

```
randReccomenderMatrix =

     2     7     2     8     8
     6     4     2     0     6
     4     5     7     7     8
     0     3     0     6     6
```

Figure 20: Initial Matrix

First off, the initial recommendation matrix must be present. Typically, there will be missing or unknown elements as well, however, in this demonstration, the matrix will be a randomized matrix with the number of users for the rows and the number of movies for the columns.

Ideally, a machine learning algorithm can factorize the matrix into two submatrices with minimal error at maximum efficiency (Grover). However, the most basic method to factorize the recommendation matrix is brute force. By running the algorithm many times, with up to 100,000 iterations, we can produce results that are in the right direction.

```
bestUserPreferenceMatrix =

     0     1
     1     0
     1     1
     0     0

bestMovieGenreScoreMatrix =

     2     4     4     3     4
     2     1     3     4     4

bestResultSoFar =

     2     1     3     4     4
     2     4     4     3     4
     4     5     7     7     8
     0     0     0     0     0
```

Figure 21: Resulting Matrix

```
comparisonMatrix =

  4×5 logical array

   1   0   0   0   0
   0   1   0   0   0
   1   1   1   1   1
   1   0   1   0   0
```

By comparing the initially recommended matrix to the resulting matrix, the resulting comparison matrix is formed in figure 22. Each 1 indicates that the resulting matrix element matches the initial element, and each 0 indicates that there is a mismatch. With this algorithm running 100,000 iterations, we were able to obtain a factorized matrix with 9/25 matching elements.

Figure 22: Comparison Matrix

Main while loop (Randomizing Matrix):

```
while(counter ~= 100000)

    randUserPreferenceMatrix = randi([0 1],numOfUsers,numOfGenres)

    randMovieGenreScoreMatrix = randi([1 4],numOfGenres,numOfMovies)

    result = randUserPreferenceMatrix * randMovieGenreScoreMatrix;

   counter = counter + 1

   randReccomenderMatrix

   result

   %this comparison matrix is a zero matrix that fills up with one's when
   %element <i,j> of both the initial matrix and the resulting matrix are
   %the same. The more ones there are, the more "similar" the matrices are
   comparisonMatrix = result==randReccomenderMatrix

   %this if statement counts up the numbers of ones in the comparison
   %matrix, and therefore, records the most similar matrix throughout the
   %loop
   if ( (sum(comparisonMatrix,'all') > sumOfComparisonMatrix) )
       sumOfComparisonMatrix = sum(comparisonMatrix,'all')
       bestResultSoFar = result;
       bestUserPreferenceMatrix = randUserPreferenceMatrix;
       bestMovieGenreScoreMatrix = randMovieGenreScoreMatrix;
   end


end
%The overall results
finalComparisonMatrix = randReccomenderMatrix==bestResultSoFar

randReccomenderMatrix

bestUserPreferenceMatrix

bestMovieGenreScoreMatrix

bestResultSoFar
```

Each iteration of the while loop randomizes two potential factors of the potential recommendation matrix and multiplies them together to create a resulting recommendation matrix. This resulting matrix is then compared to the initial matrix, and the best comparison is stored and overwritten each time.

Upon running the loop for multiple iterations, the initial matrix is compared to the best matrix created from the while loop. The resulting matrix is the most similar factorized matrix found by the while loop.

Conclusion

As technological advances are continuously made, we increase our use of ecommerce sites. Recommendation algorithms allow these sites to offer individualized experiences, tailored to each unique user. These algorithms increase traffic to the site and thereby generate more revenue for the service. Netflix, a multi-billion dollar streaming service, uses recommendation algorithms for this very purpose. These algorithms work by collecting data from the user on each type of interaction that occurs within the site. Once they collect this data, they use one of two methods to recommend titles to the user; content based or collaborative filtering. Content based filtering uses similarities in genres, based off the user's rating, in order to deduce what the user prefers. Collaborative filtering is most often applied for new users, often referred to as a cold start problem. This method determines what is popular among geographic regions, often your neighbors. It finds genres that individuals in your geographic area likes and recommends those titles to the user.

Commented [TT4]: You could combine the first and second paragraph together

In our MATLAB example, we were able to recreate an algorithm that simulates both content-based filtering and collaborative filtering. At a very basic level, these algorithms take user input values, and they assemble three key matrices, the recommendation matrix, and its two factorized matrices, which hold information about user preferences and movie ratings based on their genres. Content based filtering takes two matrices where the information is already given to us, and multiplies them together via matrix multiplication to create the recommendation matrix.

Collaborative filtering works backwards, where the recommendation matrix is broken down into its two components, the user preference matrix, and the item rating matrix. This is done through matrix factorization, with various applications such as singular value decomposition and machine learning code.

Starting off with content-based filtering, we created a MATLAB script that takes user inputs, and builds a user preference matrix, as well as a randomly generated movie genre rating matrix, in order to create a recommendation matrix. The randomly generated movie genre rating matrix can also be replaced with a matrix of selected genre ratings, and in turn, be applicable to create concrete recommendations. One key thing to take away from the content-based filtering algorithm is that it runs into the cold start problem, where the lack of initial data leads to inaccurate or nonexistent results. The solution to this problem was to ask the user for input themselves. Once the data was collected, the program took a very basic approach by building the individual matrices and multiplying them together. Overall, the MATLAB script for content-based filtering was very successful, as we were able to reproduce our sample matrices, and create accurate results with abstract values.

Regarding collaborative filtering, the main problem to solve is the factorization of the recommendation matrix. In order to factorize the matrix, the most effective way is to approach it with machine learning code, where an algorithm "guesses and checks" and improves upon itself to maximize efficiency and minimize error. However, our algorithm simulates the most basic application of collaborative filtering, by guessing and checking matrix factors in numerous attempts to find the matrix with the least amount of error from the initial recommendation matrix. By running the algorithm for 100,000 iterations, we were able to factorize a resulting matrix that was at most, matching 9/25 of the elements in the initial recommendation matrix. However, despite these poor results, this basic form of matrix factorization was done without any advanced tools such as machine learning algorithms. While the results were in the right direction, the lack of machine learning code and higher-level algorithms concluded that our simulation of collaborative filtering was inefficient and inaccurate.

In order to extend this research, the first step is to simply increase the size of these matrices. Netflix has millions of individual users, with millions of individualized recommendations for their wide range of shows. Once the algorithms can handle larger matrices with much more information, applying machine learning code to the algorithm to minimize the amount of error will complete the problem, and the filters would be fully functional. These recommendations systems have the potential to be widely applicable to all things in life that have preference. From movie preferences to online shopping preferences, or even daily needs such as food preferences. This algorithm will benefit both the business by giving them more traffic, as well as the customer by leading them to what they want.

Works Cited

"How Netflix's Recommendations System Works." 2021. *Help Center*. Accessed April 12. https://help.netflix.com/en/node/100639.

Prasad, Ankita. 2020. "The Mathematics of Recommendation Systems." *Medium*. Level Up Coding. https://levelup.gitconnected.com/the-mathematics-of-recommendation-systems-e8922a50bdea.

Nadee, Wanvimol, View Profile, Yuefeng Li, Yue Xu, Contributor MetricsExpand All Wanvimol Nadee Publication Years2013 - 2013Publication counts2Available for, Download2Citation count0Downloads (cumulative)172Downloads (6 weeks)2Downloads (12 months)22Average Citation per Article0Average Download, and Authors: Wanvimol Nadee View Profile. 2013. "Acquiring User Information Needs for Recommender Systems." *Acquiring User Information Needs for Recommender Systems | Proceedings of the 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) - Volume 03*. https://dl.acm.org/doi/10.1109/WI-IAT.2013.140.

Cates, Jill. "How to Design and Build a Recommendation System Pipeline in Python (Jill Cates). *Youtube*, uploaded by PyCon Canada, 14 Mar 2019, https://www.youtube.com/watch?v=v_mONWiFv0k

Chong, David. 2020. "Deep Dive into Netflix's Recommender System." *Medium*. Towards Data Science. https://towardsdatascience.com/deep-dive-into-netflixs-recommender-system-341806ae3b48.

GOMEZ-URIBE and HUNT. (2015). *The Netflix Recommender System: Algorithms, Business Value, and Innovation*. Https://Dl.Acm.Org/Doi/Pdf/10.1145/2843948.

https://dl.acm.org/doi/pdf/10.1145/2843948

Bourne, M. (n.d.). *4. Multiplication of Matrices*. Interactive Mathematics.

https://www.intmath.com/matrices-determinants/4-multiplying-matrices.php

Grover, P. (2020, March 31). Various implementations of collaborative filtering. Retrieved May 08, 2021, from https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0

Iqbal, Mansoor. "Netflix Revenue and Usage Statistics (2021)." *Business of Apps*, 9 Mar. 2021, www.businessofapps.com/data/netflix-statistics/.

Kumar, Vaibhav. "Singular Value Decomposition (SVD) In Recommender System." *Analytics India Magazine*, 12 Jan. 2021, https://analyticsindiamag.com/singular-value-decomposition-svd-application-recommender-

system/#:~:text=SVD%20is%20a%20matrix%20factorisation,as%20a%20collaborative%20filtering%20technique.

Le, J. (2020, June 28). Recommendation system series Part 4: The 7 variants of MF for collaborative filtering. Retrieved May 08, 2021, from https://towardsdatascience.com/recsys-series-part-4-the-7-variants-of-matrix-factorization-for-collaborative-filtering-368754e4fab5

Serrano, Luis. “How does Netflix recommend movies? Matrix Factorization.” *Youtube*, uploaded by Luis Serrano, 7 Sep 2018, https://www.youtube.com/watch?v=ZspR5PZemcs.

Vozalis, Manolis G., and Konstantinos G. Margaritis. “Applying SVD on Generalized Item-Based Filtering.” *Caridokumen*, 28 May 2017, https://caridokumen.com/download/applying-svd-on-generalized-item-based-filtering-_5a467690b7d7bc7b7a0ade68_pdf.