What is the Value of Rating Obscure Items?

An Analysis of the Effect of Less-Popular Item Ratings on Recommendation Quality

A Dissertation

SUBMITTED TO THE FACULTY OF THE

UNIVERSITY OF MINNESOTA

BY

Anshuman Narayan

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTERS OF SCIENCE

Joseph A. Konstan

June, 2019

Dedicated in honor of

Shashi Narayan and Krishnan Adhi Narayan

Acknowledgements

Many people have helped me along this two-year journey to complete my Master's degree and I would be remiss in not thanking them. I would like to thank Professor Joseph Konstan for guiding me through this journey and teaching me so much about not only recommender systems but also how to conduct good research work and inculcating in me the perseverance to conduct and complete this work. I owe a lot of what I have learned these two years to him. I would like to thank the Professors at the University of Minnesota that I've been fortunate enough to study under for all the knowledge I've gained from them. I have gained immense respect and inspiration from witnessing the passion they hold for teaching. I leave the this University knowing so much more, and knowing that there is so much more to know, and I am indebted to them for this. I would like to thank members of the GroupLens research lab for their patience and kindness in helping me,providing me great advice and always being open to discussing whatever I was stuck with. I would like to thank the friends I've gained over the last two years for all the help they have provided me. I would like to thank my friends back in India for being such strong sources of encouragement and motivation, I would not have been able to do it without them. I would like to thank my family. Their love, support and sacrifice is the sole reason I am what I am today. Thank you for being there for me at every setback. Thank you Amma, Appa, Akka and Attimber.

Abstract

Recommender systems designers believe that the system stands to benefit from the users rating items that do not have many ratings. However, the effect of this act of rating lesser known items on the user's recommendations is unknown. This leads to asking the question of whether these low popularity items affect the recommendations received by users. This work looks at the effect less popular items have on a user's recommendations and the prediction and recommendations metrics that quantify the quality of recommendations.

Using a matrix factorization model to build a recommender system, we modify a subset of users' ratings data and look at the difference in recommendations generated. We also make use of popular recommender systems metrics such as nDCG, Precison and Recall to evaluate the effect of these modifications.Apart from looking at the effect of this "truncation" of casual user ratings data on the casual users themselves, we also look at the effects of this "truncation" on the more invested users of the system, in terms of top-n recommendation and prediction metrics. The results of these evaluations appear promising, with very little to no loss of information, personalization or metric scores for more casual users. The results of these evaluations for more serious users also appears to have little effect on the performance of top-n recommendation and prediction metrics.

Contents

List of Tables **v**

List of Figures **vi**

1 Introduction **1**

2 Recommender Systems Algorithms and Interfaces **3**

2.1 Recommender Systems Algorithms 3

2.2 Recommender System Interfaces 8

3 Research Questions and Experimental Setup **11**

3.1 Research Questions 11

3.2 Offline Experiment Setup 14

3.3 Recommendation Model 17

3.4 Experimental Setup 21

4 Offline Experiment and Results **22**

5 Conclusions **37**

Bibliography **39**

List of Tables

3.1 RMSE Calculation for Feature Selection 20

4.1 nDCG,Precision and Recall of Casual Users Test Sets 33

4.2 5 Crossfold RMSE Test Results 33

4.3 RMSE Scores of Casual Users Test Sets 34

4.4 RMSE Scores of Serious Users Test Sets 35

4.5 Top-N Recommendation Metrics on Serious Users Test Sets 36

List of Figures

3.1 Distribution of Movies by Popularity(on a Logarithmic Scale) 15

3.2 Histogram of Percentage of Popular Movies rated by Users 16

4.1 Histogram of Cosine Similarity between Full and Popular Only User Feature Vectors(n=1000) 23

4.2 Histogram of Cosine Similarity between Full and Popular Only User Feature Vectors Grouped by Percentage of Movies Rated That are Popular(n=1000) . . 24

4.3 Histogram of Number of Popular Movies Recommended to Full User Profile vs Popular Only User Profile(n=1000) 27

4.4 Plot of Number of Top-20 Popular Movies found in Recommendations to Full User Profile vs Popular Only User Profile(Total number of users=1000) 28

4.5 Comparative Histogram of Diversity in Recommendation List between Full User Profile and Popular Only User Profile(n=1000) 29

4.6 Histogram of Overlap Between Recommendations given to Full User Profile vs Popular Only Profile 30

Chapter 1

Introduction

Recommender systems use the information gained from it's users to generate recommendations. Recommender systems employ a variety of methods to gain user preference including explicit methods such as having users rate items or implicit methods such as seeing which items users interact with. Designers of such systems however face a problem in the distribution of these ratings or information across items. It is often observed in such systems that the ratings follow a long-tail distribution. Some of the items accrue most of the ratings while there are a wide number of items that do not have as many ratings. System designers believe that if these items were to receive more ratings, then the recommender system model would have more information about the items in it and provide more confident recommendations on these items to its users. However, it is unknown whether users really benefit directly in their recommendations if they rate such lesser known items. In this work, we explore what effect these lesser known items have on the user recommendations and what value they provide to the model that is generated by the users. We look at what effect does removing less popular items have on the recommendations of casual users. We also follow this by looking at the effect that the removal of these ratings by casual users have on other users.

Chapter 2 provides some insight into the difference recommender system algorithms developed to generate recommendations for users. Chapter 3 explains the research questions

posed in this offline experiment and the experimental setup. Chapter 4 provides the methodology of the experiments themselves and the results generated from them. Chapter 5 provides the conclusion drawn from these results.

Chapter 2

Recommender Systems Algorithms and Interfaces

2.1 Recommender Systems Algorithms

The simplest way to define a recommender system would be any computer system that provides users with suggested items from a set of items based on what the system thinks the user likes. These suggestions could range from what movie to rent, what television series to watch next to what kind of headphones they would like to use. Recommender systems serve as a solution to the problem of many user when they encounter a service that provides them a wide variety of options to choose from. These systems serve to filter through most of what the user is likely to be uninterested in and pick out what they would like. Such a system would have to predict from its set of items what a user would like and present them to the user.

Recommender systems generally consist of the following components. They have a set of users for whom they generate recommendations, a set of items that are to be recommended to the users, and information about the nature of the items or the preferences of the users for those kind of items. Using this information, recommender systems can generate a ranked list of items for each user that it believes contains items the user might be interested in. The information that the system has about the users and the items in the system can either be content based information(if we considered items as movies, the genres that define the movie,

the director, the cast etc. and information about what kind of movies users like to watch) or information about the user's preference for certain items in the form of ratings(explicit ratings) or data about which items users purchased or consumed(implicit ratings). Using this data, the system can employ a variety of techniques to generate recommendation for it's users. The two most popular categories of recommender systems are content based filtering techniques and collaborative filtering techniques.

2.1.1 Content Based Filtering Techniques

Content Based Filtering Techniques are a subclass of recommender system algorithms that use descriptions for the items and the user preferences for the items to understand user taste and recommend new items to the user. Generally, such techniques often involve trying to clean existing data about each item, looking at the subset of items that a user elicits preference and summarizing the data about these items to form a user vector and then using this user vector as a yard-stick to score the other items in the system and recommend the items that score the highest for that user. This chapter[15] from the Recommender Systems Handbook by Ricci et al.[18].

2.1.2 Collaborative Filtering Techniques

Collaborative Filtering Techniques overcome a huge shortcoming of content based filtering methods. Content based filtering methods depend heavily on the existence of some descriptive data about the items in the system, which may not be readily available always. Also, it is possible that this descriptive information does not capture all the properties of the items that could lead to the user liking them. Collaborative filtering techniques instead leverage the collective knowledge gained from the preference elicitation of all users to make recommendations. In collaborative filtering, users are grouped together based on their rating

behaviour. Two users are considered similar if they both rate the same items same values. Each of these groups are called as a neighborhood for any user. The recommendations are then generated from the items that users in the neighborhood rate positively which the user being recommended for has not rated. This concept of collaborative filtering can also be extended to building neighborhood for items. In this case, we generate recommendations for a user by looking at the items in the neighborhood of the items they have already rated. Collaborative filtering techniques have been one of the more well-explored spaces of recommender systems. We will now provide a brief look into the history of collaborative filtering and how the techniques have developed into algorithms that are now used widely in the industry.

2.1.3 Advances of Collaborative Filtering

The first mention of the method of collaborative filtering, which involved leveraging the preferences of users to find like-minded users whose ratings could be used to generate recommendations, was first observed in Goldberg's Tapestry paper[8].The Tapestry system, was an email filtering system that generated custom email lists for users by encouraging users to annotate their emails and use this annotations when querying emails to generate email lists for users. This paper formed the foundation of collaborative filtering, an technique that was widely explored in the fields of recommender systems and information retrieval.

While this method of collaborative filtering was manual, in that users choose who's activity to follow, there was further work done in the field of automating the collaborative filtering. Herlocker's paper[11] outlined the basic framework of a collaborative filtering algorithm including the methods employed for calculating the similarity score between users, how the neighborhood is selected, what the size of the neighborhood should be and how the

prediction for an item is generated after the neighborhood is decided. This paper provides a variety of approaches at each stage and provides comparative data about these different approaches.

Herlocker's paper generally looked at a user-user neighborhood model.The second type of collaborative filtering models had a item-based solution as detailed in Sarwar et al.'s paper[20] which describes a class of item based collaborative filtering algorithms. In this model, neighborhoods of item's are built and recommendations are generated from a pool of candidate items that are neighbors to the items the user has rated.

While collaborative filtering techniques became more and more popular, the efficiency of neighborhood methods for larger systems were questioned. Calculating similarity scores for a large pool of users and selecting a portion of them and looking at their ratings to generate predictions for a single user and item as an operation did not scale well. It was at this time that the technique of dimensionality reduction was proposed as a solution to this problem. Sarwar et al.'s report[19] looks at applying Singular Value Decomposition(SVD) on the ratings matrix. SVD generates three factor matrices that when multiplied reproduced the original matrix. These three matrices can be used to generate a lower dimension approximation of the original matrix. Since ratings data is sparse, as most users have not rated most items, dimensionality reduction allows us to represent this matrix with a lower dimension dense matrix. The technique of SVD can only be applied to dense matrices themselves. This issue is solved in the report by considering the average rating of a user and an item to fill-in for the user, item pair that does not have a rating.

Other methods were proposed to solve the fill-in issue, including Goldberg's Eigentaste paper[9] which considered only a subset of a items that all users have rated and building a lower dimension model from this data using a dimensionality reduction technique called Principal Component Analysis(PCA).

As a result of the Netflix prize, a competition where Netflix released some of it's users data and challenged anyone to outperform there existing algorithm's performance on root mean square error(RMSE), Simon Funk devised FunkSVD[7]. FunkSVD is a method of arriving at the rank-k approximation of the sparse ratings matrix without any filling in any of the zeroes in the sparse matrix. This is done by iterating over the given ratings, training a single feature or reduced dimension by trying to reduce the approximation error and repeating this process over the number of reduced dimensions desired.

Following FunkSVD, many advances were made in improving matrix factorization methods which are summarized in Koren et al.'s article [14] and are surveyed in further detail in this chapter of the Recommender system handbook [13]. One of the bigger advancements included having algorithms that scaled well by computing the reduced matrix factors in parallel. The most common method employed to do this is called the Alternating Least Squares method or ALS, which alternatively solves for two lower dimension matrices that try to reduce a loss function which is based on the squared difference between the known ratings and the predictions generated by these lower dimension matrices. ALS was first used in recommender systems by Bell et al. [1] where they used the technique to quickly learn the weights of neighboring items in a recommender system. Zhou et al.'s method [23] used the ALS approach to scale the matrix factorization problem so that the each of the features of the reduced dimensional space could be derived independently. They use a weighted regularization in their loss function, which is reduced to find the best values for each feature, to avoid over-fitting. The individual lower dimension matrices are generated by factorizing over the known ratings and the regularization term ensures that the trained columns of the matrices are not over-fitted.

2.2 Recommender System Interfaces

Apart from looking at the what could be done to improve the quality of recommendations for a user, there exists a whole field of study that looks at the interface of the recommender system and the changes that can be made to the interface that users experience when engaging with the recommender system to improve user satisfaction. This work includes looking at what data about each item is displayed, what rating scale is used to elicit user preference among others. Cosley et al's paper [5] is an example of such work, where different ratings scales are analyzed and the effect of showing displaying predictions when users rate an item. There has also been a lot of work done to use different interfaces to learn user preferences and better serve recommendations to users.

While most recommender systems followed the paradigm of providing users with a whole set of recommendations at once, conversational recommenders mimic the method by which real-life recommendations are generated,Shimazu et al.'s paper[21], a conversational agent that alternated between proposing items and asking about the proposed items as a way to naturally navigate the item space and arrive at some recommendations, is an example of an alternative approach to recommender systems.Similarly, Burke et al's paper[4], which proposed FindMe systems, tour guides of digital catalogs of products to arrive at a product they can confidently recommend to a user, also served as significant contribution towards conversational interfaces.

In a lot of these models, there is no prior knowledge about the users preferences and these are learned over the course of the dialog between user and the recommending agent. Case-based recommender systems employ a similar technique. User's provide a variety of preferences on the item they wish to be recommended. The model looks up existing cases of items that match the preferences of the user and then return the most similar items

as recommendations. Bridge et al's paper [3] describes a framework for such case based reasoning systems.

Reilly et al.'s [17] paper talks about dynamic critique systems that built on the "FindMe" navigation paradigm, where the feedback that can be provided by the user is static(hence static critique systems), which can dynamically change how a user can navigate the interface depending on item availability and user preference.

2.2.1 New User Problem

Recommender systems face the issue of trying to generate recommendations for new users. Since recommendations are heavily dependent on what the system knows about the user's taste to ensure that the recommendations are personalized, it can be difficult for such models to ensure that they generate good enough recommendations for the user without having a lengthy sign-up process. Rashid et al.'s paper[16] explores what items are best suited to learn a user's preference and proposes multiple approaches of asking users to rate items with varying degrees of control afforded to the user. They learn that users prefer to have more control over the items they would like to elicit a rating for even if the process takes longer. They also suggest that user's prefer to rate more popular items as they are more likely to have an opinion about them. Jones' paper [12] tells us that a simple, easy to understand interface that requires a low amount of initial effort from users and good quality initial recommendations can make a difference to how invested user's can be in the system.

An interesting aspect of recommender systems is the idea of information overload. Providing new users with many recommendations can lead to indecisiveness of the user. Too many recommendations can lead to users being unable to decide which item to consume. Bollen's paper [2] on choice overload talks about how having too many good recommendations can be detrimental to user satisfaction, especially if they are new users.According to

this paper, having a diverse set of recommendations that exceed no more than 20 in number is ideal to avoid the issue of choice overload. Willemsen's paper [22] talks about using the latent factor model to ensure that recommendations are diverse to ensure that user's can make easier decisions about choosing items from a set of recommendations.

Chapter 3

Research Questions and Experimental Setup

The purpose of this study was to look at the effect of limiting the ratings of a user in a recommender system to a reduced fraction of the total ratings by considering just the popular items in their profile. We believe that there exists simpler user interfaces for casual users of a recommender system that limit to just the popular items in the system. We believe that such a reduced interface is easier for a user to navigate and given it's simplicity is more fulfilling to use. However, we were worried that removing a subset of these ratings could limit the ability of user's to rate more obscure items which would hurt the user's ability to express their personal taste and prevent users from receiving high-quality recommendations. We thus designed an offline study to answer these concerns. We posed the following questions that drive our study to assess the impact of limiting a user's profile to just popular items.

3.1 Research Questions

To address our concerns, we formulated the following questions:

Q1. How much do user models, expressed as feature vectors, change when profiles are truncated to only more-popular items?

We wish to know what effect do these less-popular ratings have on the perception our model has of the user. Since the recommendations generated to a user are affected by the feature vector that the model generates for them in the lower dimension space, we wish

to know if the ”less-popular” items have an affect on this representation of the user in the model.

Q1a. Does this change vary with the percentage of a user's total ratings that are among the popular items?

We wish to understand if there is any marked difference between users who have more ”less-popular” items rated on their profile and users who have more ”popular” items rated on their profile when we remove ”less-popular” items from their ratings. The question we are asking here is do profiles change more if a user has more ”less-popular” items rated that would get removed when truncating?

Q2. How are the top-n recommendations produced for users with truncated profiles different from those generated for the same users with their full profiles?

We wish to evaluate the differences in recommendations that a user receives when the model generates a feature vector on all of their ratings(”Full”) vs when it generates a feature vector on only their ”popular-only” profile. We wish to in what ways recommendations are affected by ”truncating” the user profile.

Q2a. Are the top-n items recommended more or less popular for the truncated user profiles?

Removing ”less-popular” items from a user's rating may cause an issue where these users do not receive ”less-popular” item recommendations. This question is interested in looking at if rating an ”less-popular” movie necessarily leads to being recommended ”less-popular” items.

Q2b. Are the top-n item recommendations more or less personalized for truncated profiles?

Since removing ratings from a user's profile would reduce the amount of information we know about a user, we would like to see if recommendations become more generic and

less personalized if we ”truncate” our users’ profiles.

Q2c. How much overlap is there between the top-n recommended items for full and truncated user profiles?

We wish to know how many of the recommendations a user receives are unaffected by the ”less-popular” items that they rate. We would like to know if users are recommended the same items even if we lose ”less-popular” movie ratings from their profile.

Q3. What is the effect of removing these ”less-popular” items from a user’s profile on the top-n recommendation metrics?

Recommendation metrics are used to gauge the performance of the algorithm. We wish to see if removing ”less-popular” items affects the different top-n recommendation metric scores that are generated for a user.

Q4. How does truncating a user’s profile affect the accuracy of item predictions?

The ability of a recommender is correctly predict the user’s preference for an item is an important attribute to measure. We wish to know if there is really any information loss for the model’s understanding of a user in terms of it’s ability to accurately predict ratings by removing ”less-popular” items.

Q4a. Is the change in accuracy different for predictions on popular vs. less-popular items?

We want to understand if the effect of removing ”less-popular” items from a user’s profile affects the prediction on ”popular” items which would tell us that we are indeed losing information about our user by removing the ”less-popular” items.

Q5. Does truncating ”casual” users profiles to ratings of only ”popular” items affect the performance of the system for ”serious” users?

Most of our questions address the affects of ”truncating” on the user themselves. We are also, interested in seeing if these ratings(of ”less-popular”items by ”casual” users) have

a role to play in the recommendations and predictions of other more serious users. We wish to understand if the ratings of ”less-popular” items add meaningful value to the recommendations received by more serious users. We design our experimental setup and analyze the predictions and recommendations generated from our model to provide answers to these research questions.

3.2 Offline Experiment Setup

For the purposes of this experiment we use the MovieLens 20 Million ratings data-set[10]. This data-set consists of 20 million ratings from the MovieLens website, a movie recommendation website run by the GroupLens research group at the University of Minnesota where users sign up into the system, provide ratings to the movies present in the system and receive recommendations from one of many recmmender algorithms that they choose from. The website also serves to host various online experiments conducted by members of the research group. The data-set used for this evaluation is publicly available to download from the Grouplens website. It consists of 20,000,463 ratings provided by 138,493 users on 26,744 movies. The data-set contains only user and movie ids along with a rating for each user item pair. Like most data-sets generated from online recommender systems, this data-set is also long-tailed i.e. there exists a small number of movies that have most of the ratings whereas a large portion of movies have very few ratings. This usually occurs because users are more likely to rate items that they are already aware of(usually popular movies) and the system is more confident in recommending these movies to its users.

Since popularity has played an important role in defining our questions, we first begin by defining what popular movies are in this dat-aset. We define a movie as popular if it belongs to the top 2500 movies in the database by number of ratings. A movie is said to be

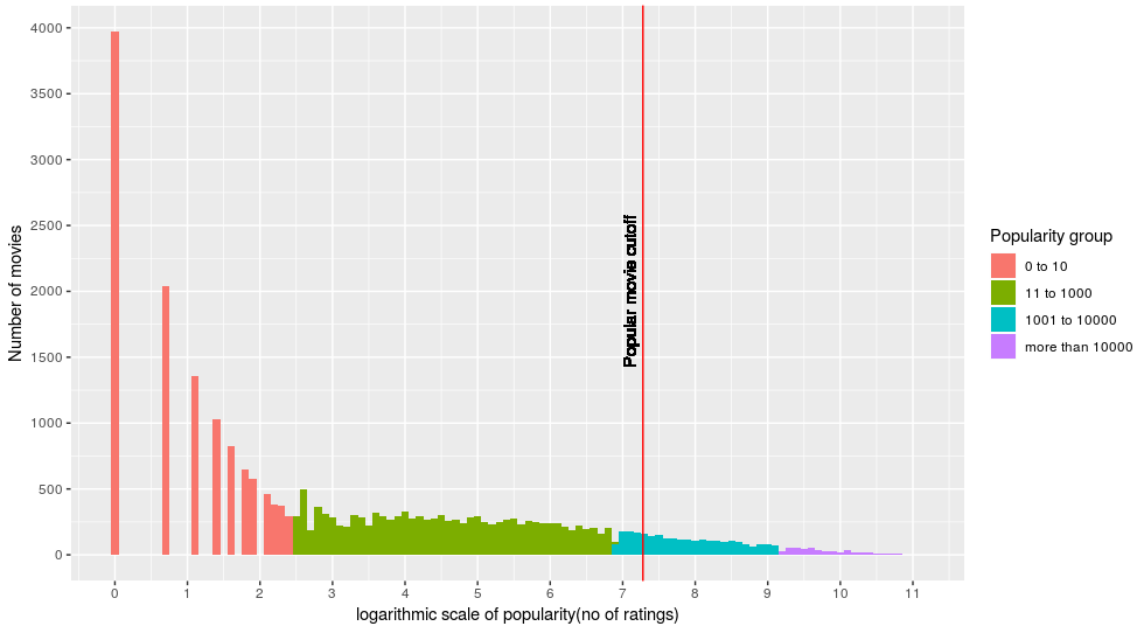

Figure 3.1: Distribution of Movies by Popularity(on a Logarithmic Scale)

in the Top 2500 if it has at least 1450 ratings as the number of ratings a movie can be called its popularity. When looking at the distribution of ratings across movies, we noticed that the Top 2500 movies account for approximately 16.9 million ratings of the 20 million ratings in the data-set. Top 2500 movies account for 85 % of the total ratings in the data-set. For our experiment, we define popular movies as those that are in the Top 2500 movies by popularity. Movies outside this group we define as unpopular. To visualize the distribution of ratings by movies we plot a histogram of the log popularity of movies. With this histogram we can visualize how few our Top 2500 movies are compared to all the movies in the database and how skewed the number of ratings are for this small group of movies.

The red line in Figure 3.1 indicates the popular movie cutoff. Bars to the right of this line represent the Top 2500 movies or popular movies.

Onto our users, we need to define our users into groups to understand which group to run our evaluation on. We want to run the evaluation on a group which has just started

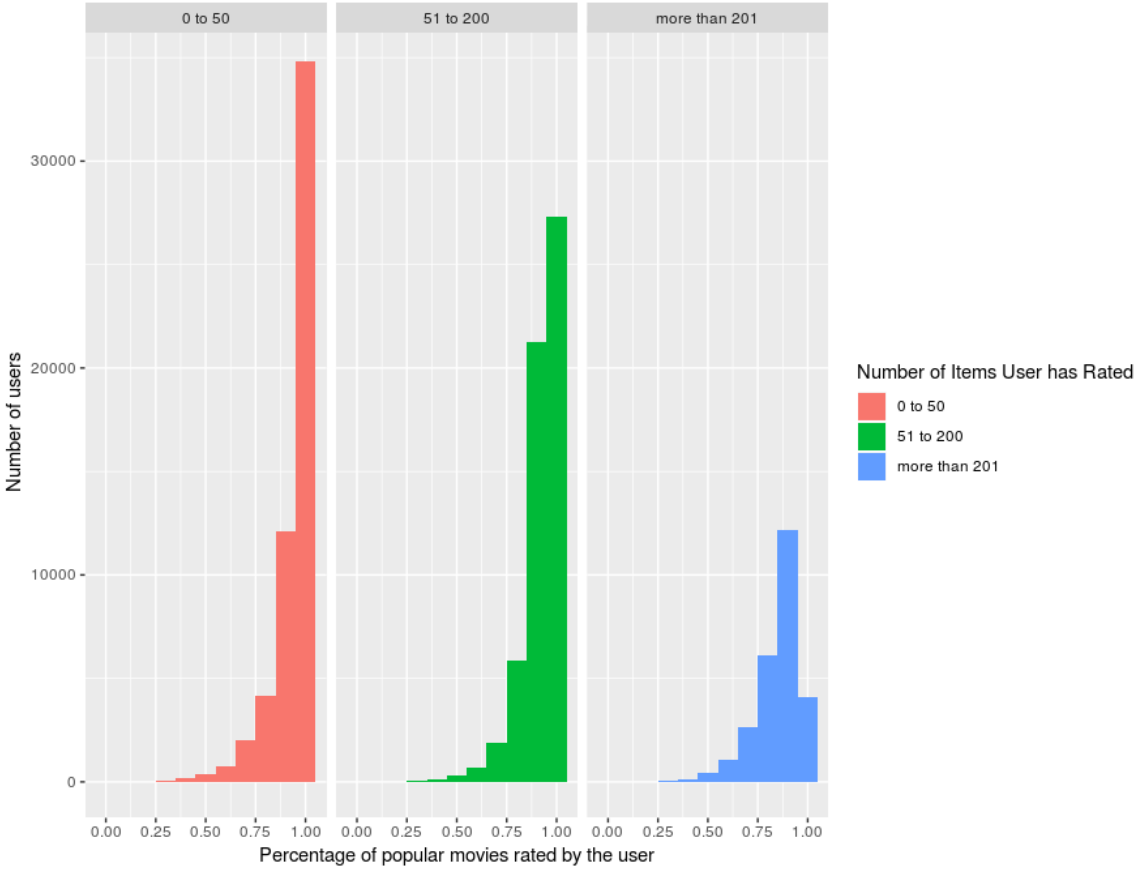

Figure 3.2: Histogram of Percentage of Popular Movies rated by Users

rating unpopular movies,since that would allow us to look at what does effect does rating unpopular movies have on a user's recommendations. We first try to understand how many users rate unpopular items. To do this, we calculate for each user, the number of their movie ratings that have been provided to popular movies by the total number of movies that they have rated.

We plot this measure for each user in a histogram. We divide users into 3 groups. Users who have rated between 0 to 50 movies, users who have rated between 50 to 200 movies and users who have rated more than 200 movies. We see the histogram of this measure for each of our groups in Figure 3.2 . As expected, users who rate between 0 to 500 items

rate mostly popular movies. Users in the 50 to 200 item group rate a larger percentage of unpopular movies than the first group. For our offline evaluation, we consider the users in the 50 to 200 groups as candidates for our evaluation. This is because we believe that for our system, rating at least 50 movies as users who have invested in the system beyond the initial sign-up process.These users have some amount of investment in the system. At this point, we believe that they have explored the popular movies domain and are entering the unpopular movie space. We term these users as ”casual” users who have just begun an investment in the system and we consider the users who rate more than 200 movies as ”experienced” users. We cap our candidate set of users who have rated 200 movies because these ”experienced” users have rated a wide variety of popular and unpopular movies. In our evaluation we are more interested in looking at what makes ”casual” users more invested in the system.

3.3 Recommendation Model

For the purposes of this study we decide to employ a matrix factorization based collaborative filtering model. We do this for two reasons, first , matrix factorization models are the most commonly used techniques to generate recommendations on online platforms. Second, matrix factorization generates for us a model that represents each user and item in the system. It is thus easier for us to answer our questions by employing this model. For this evaluation, we choose the Alternating Least Squares factorization method. Specifically the algorithm that we implement is called Alternating Least Squares with Weighted Regularization, as proposed in this paper [23] by Zhou et al.

In this method, for a given Matrix of ratings R of size $m * n$ where m refers to the number of users and n refers to the number of items, where each of the ratings in R is defined as $r_{i,j}$

where i, j refers to user i and item j, we generate two lower-dimension matrices U of size $m * k$ and M of size $n * k$ where k is the lower dimension value of our choosing. It follows that $k << min(m, n)$ and the matrices U and M form a lower order approximation of our original R matrix. We refer to the matrix U as the "user" matrix that consists of m vectors of length k that are the representations of each user in this reduced dimension space.

The algorithm initializes one of the matrices with random values. Let us assume this is done for the M matrix. Now, keeping the M matrix constant, we find the U matrix that minimizes the following loss function, as mentioned in [23]:

$$f(U, M) = \sum_{i,j \in I} (r_{i,j} - u_i^T m_j)^2 + \lambda \Big(\sum_i n_{u_i} ||u_i||^2 + \sum_j n_{m_j} ||m_j||^2 \Big) \tag{3.1}$$

where I refers to an index set of all known ratings in the system. n_{u_i} refers to the number of ratings that the user i has and n_{m_j} refers to the number of ratings that have been provided for item j. Here u_i refers to the column i of the user matrix U and m_j refers to the column j of the item matrix. These can also be described as the user feature and item feature vectors for the respective user and item.

Once a suitable U matrix is found that minimizes the Equation 3.1, we then fix the u matrix and solve for the M matrix that minimizes our loss function. This process is repeated till the models converges.

These algorithm can be summarized in the following steps:

Step 1 . Initialize the item matrix M with the average rating of each item in the first row and small random entries for the remaining rows.

Step 2 . Keeping the matrix M constant, solve for the matrix U that minimizes the loss function.

Step 3 . Keeping the matrix U constant, solve for the matrix M that minimizes the loss function.

Step 4 . Repeat Steps 2 and 3 till a stopping criteria is satisfied.

In Step 2, finding the U that minimizes the loss function in Equation 3.1 is done by calculating each vector u_i in the matrix U from the following formula:

$$u_i = A_i^{-1} * V_i, \forall_i$$

where $A = M_{I_i^U} M_{I_i^U}^T + \lambda n_{u_i} E$ and $V_i = M_{I_i^U} R^T(i, I_i^U)$ and E is an $n_f x n_f$ identity matrix. I_i^U is the set of indices of items that user i has rated. $M_{I_i^U}$ refers to a sub-matrix of the item features for the items that the user i has rated.Similarly,$R(i, I_i^U)$ is a row vector of original ratings matrix R where only ratings that the user has provided(given by I_i^U) are selected.

Similarly in Step 3, finding the M that minimizes the loss function in Equation 3.1 is done by calculating each vector m_j where we update the values by using the user features of the users who rate item j. The update is given by the following formula:

$$m_j = A_j^{-1} * V_j, \forall_j$$

where $A_j = U_{I_j^M} U_{I_j^M}^T + \lambda n_{m_j} E$ and $V_j = U_{I_j^M} R(I_j^M, j)$. I_j^M is the set of indices for all user who have rated item j. $U_{I_j^M}$ is the sub matrix of the user matrix U that has only the user feature vectors of the users that have rated the item j. $R(I_j^M, j)$ is the column vector of ratings of the item j, with values only in the rows of the users who have rated the item.

These updates are repeated until the stopping criteria is reached. In our case, this is defined if the number of iterations have been reached.

With the algorithm defined, we proceed to find the best parameters for our model. The implementation of the algorithm we are using here is from the LKPY python package[6]. For this algorithm, we need to tune two parameters for the algorithm, the regularization rate and the number of features and the number of iterations to train over(our stopping criteria). We first try to determine the number of features that would work best for our algorithm. We

Algorithm with No. of Features	RMSE Score
ALS-10	0.81484
ALS-15	0.81416
ALS-20	0.81405
ALS-25	0.81402
ALS-30	0.81400
ALS-40	0.81399
ALS-50	0.81399
ALS-60	0.81398
Bias Based Scorer	0.8598

Table 3.1: RMSE Calculation for Feature Selection

do this by running a 5-fold Cross evaluation of RMSE of the model on the full data-set. We run this evaluation multiple times by varying the number of features and using the default values for the regularization rate and number of iterations. We also evaluate the RMSE for a biased scorer model which predicts for a user item pair by adding the average rating the user gives, the average rating of the item receives and the global average of all ratings in the system. This model is used as a baseline for the scores generated by our candidate models.

Table 3.1 lists the Algorithms(and the number of features in case of our candidate models) and the average RMSE score.From the resulting table we can see that having more than 25 features does not reduce the RMSE significantly. We thus finalize on using 35 features to represent our data as a reduced dimension model.

To finalize of the regularization rate, we looked at the model with 35 features and for each user in the system, we generate the largest and smallest prediction. We do this to ensure that this range approximately fits our rating scale of 0 to 5. We found that using the regularization rate of 0.1 and increasing the number of iterations to 50 ensured that for each user our range of predictions ranged approximately between 0 to 5. We thus finalize on the parameters of our model with 25 features, trained over 50 iterations with a regularization

rate of 0.1.

3.4 Experimental Setup

Having prepared our data set and finalizing on the parameters of our recommendation model, we define our test setup. We consider a sample of a 1000 users from our group of ”casual” users. We sample users who have rated at least one ”unpopular” movie. We generate a copy of the ratings of our test users. For this copy, we remove the ”unpopular” movies from the ratings. These ratings help form the truncated model of our test users. We train our model on the 20 million ratings set and train new user profiles of the truncated user ratings on the existing item matrix of our model. With this we have our offline experiment set-up to perform tests to answer our research questions. In our evaluations, we will analyze two set of the ”test” user profiles. The first, the profiles for the user generated from all their ratings in the original model, which we refer to as the ”Full” profile vector. The second, the profiles for the user generated from only their ”popular” movie ratings, which are generated by folding in these ratings to the existing model, which we term as ”Truncated” user profiles. The evaluations performed during this work only look at the recommendations and predictions generated on these 1000 ”casual” users.

Chapter 4

Offline Experiment and Results

This chapter will list the research questions posed in Section 3 and will describe how the question in each case was answered in the offline evaluation.

Q1. How much do user models, expressed as feature vectors, change when profiles are truncated to only more-popular items?

After training the model, we have access to the user features matrix which contain the feature vectors,or user models, of the "full profile" of our users. The result of the our folding-in operation with the truncated ratings data(containing the ratings of test users with only their "popular" movie ratings) gives us another user matrix that contains the user feature vectors, or user models, of the "truncated profile" of our test users. We iterate over our 1000 test users and extract the "full" and "popular-only" feature vector for each user. We then calculate the cosine similarity between these two profiles of a user. The cosine similarity for two vectors u and v can be formulated as the following:

$$\frac{u * v}{||u||_2 * ||v||_2}$$

We calculate the cosine similarity between the "Full" and "Truncated" profiles of each of the users and store these values. We then plot a histogram of these values.

This histogram is displayed in Figure 4.1 . To provide a baseline of the similarity between users in the test set, we look at the average pairwise cosine similarity between "test" users. We calculate this average pairwise cosine similarity between for both the "Full"

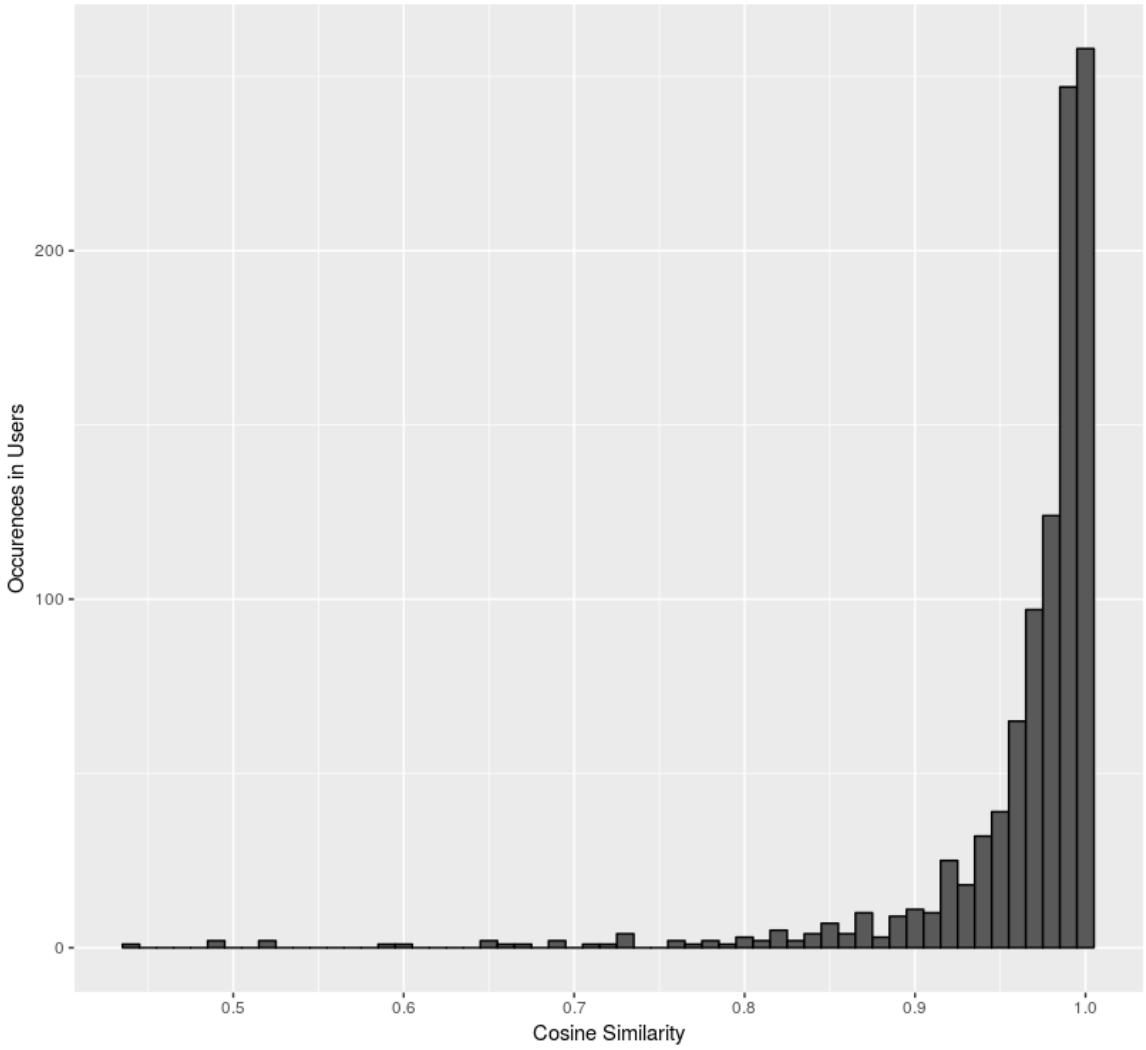

Figure 4.1: Histogram of Cosine Similarity between Full and Popular Only User Feature Vectors(n=1000)

profile and the ”Truncated” profile.The average pairwise cosine similarity between ”Full profiles” is 0.00035. The average pairwise cosine similarity between ”Popular only” profiles is 0.0010.The average cosine similarity between any two users in the system is 0.008. This tells us that, the users in the our selected ”test” set do not have similar taste and lead us to believe that the as the histogram suggests, most users are unaffected when they are truncated to ”popular” movies only profiles.

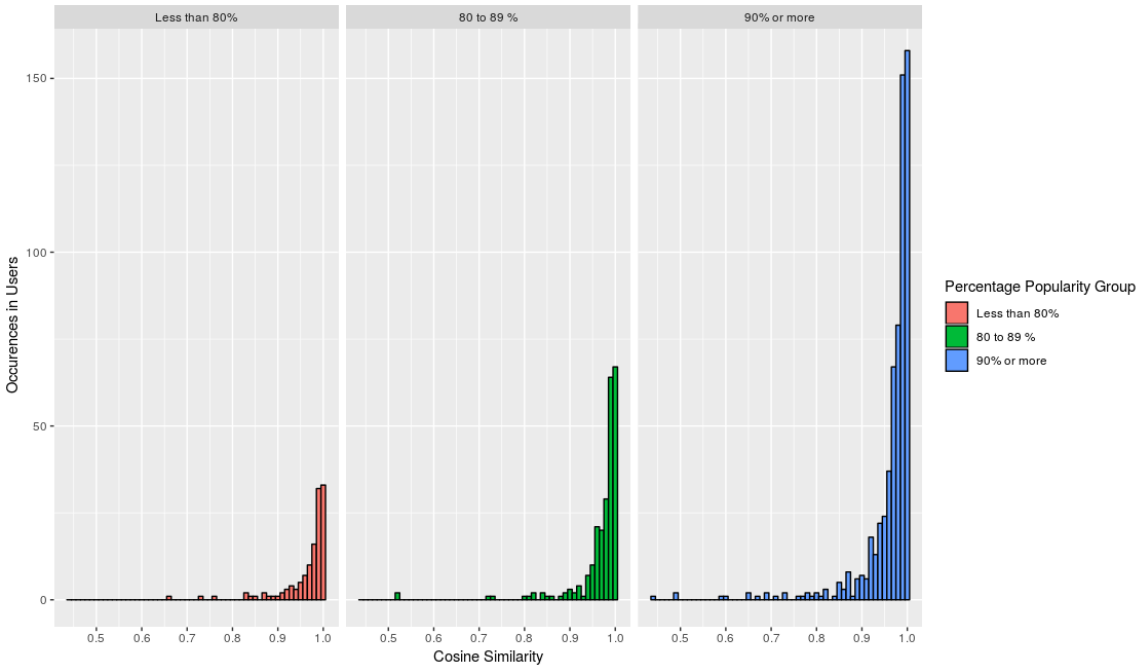

Figure 4.2: Histogram of Cosine Similarity between Full and Popular Only User Feature Vectors Grouped by Percentage of Movies Rated That are Popular(n=1000)

Q1a. Does this change vary with the percentage of a user's total ratings that are among the popular items?

With our cosine similarity scores, we align the scores with percentage of a user's total ratings that are among popular items. We had previously calculated this information to generate the histogram in Figure 3.2 to look at what percentage of a user's ratings consist

of ”popular” movies by different user groups. We recreate the cosine similarity histogram for each of these groups of users(users who rate between 0 to 50 movies, 50 to 200 movies, more than 200 movies).

In this histogram in Figure 4.2, each of these groups do not have the same number of users in them. This is because most ”casual” users have 90 % or more of their ratings as ”popular” movie ratings. However, we did notice that having more ”unpopular” movies in the rating from does slightly decrease the similarity between user profiles.

Q2. How are the top-n recommendations produced for users with truncated profiles different from those generated for the same users with their full profiles

Having generated our user and item matrices from training our model, we analyze the recommendations generated for top-20 recommendations. We generate the recommendations in the following manner, we first set up our candidate items for recommendations. We do this taking at the movies in the system and removing the movies that a user has already rated. We then remove any items that have less than 10 ratings as we cannot confidently recommend these items. We then predict the rating the user would give to the remaining items. This is done by calculating the dot product of the user vector with the item vector and adding the user, item and global bias. Then, these predictions are ranked in descending order. The largest 20 predictions are the Top-20 recommendations for that user. We repeat this process for our 1000 test users' ”Full” profile and ”Truncated” profiles.

Q2a. Are the top-n items recommended more or less popular for the truncated user profiles?

To assess the popularity of the recommendations, one's first idea would be to look at the average popularity(no of ratings) over all the 20 items recommended to a user. However, the issue with this method is that if one of the items is very popular(say has 20,000 ratings) and the remaining items have a popularity of 1 each, the average would not paint the right

picture about the overall popularity of the recommendation list. Instead, we look at our definition of ”popular” movies to measure the popularity of our recommendation list. We look at how many of the movies recommended to a user overlap with the Top 2500 most popular movies. We calculate this overlap for the ”Full” and ”Popular” profile. We then plot this data in a comparative histogram of the overlap of a recommendation list with the Top 2500 most popular items.

From looking at the histogram in Figure 4.3 we understand that on average, the recommendations have the same number of Top 2500 movies if we use a ”Full” profile or a ”Truncated” profile. We see that removing the ”unpopular” movies does not increase the number of ”popular” movies that are recommended to a user.

Q2b. Are the top-n item recommendations more or less personalized for truncated profiles?

To measure the level of personalization of a user's recommendation, we consider a baseline of recommendations that are least personal. This baseline, in this case, would consist of the 20 most popular movies in the data set. These are the movies that a recommender would give a user if it knew nothing about them. To measure if ”Truncated” profile is more or less personalized, we look at the overlap of a test user's top-20 recommendations with the Top-20 most popular movies. We calculate this overlap for both the ”Full” profile as well as the ”popular only” profile. We then plot the overlap of a user's ”Full profile” against the overlap of a user's ”Truncated” profile. In our case, since a lot of the data points cluster at specific values, we denote an over-clustering of points with a larger circle and provide a legend to show what the size of the circle can tell us about the number of points that are being plotted at that point.

As we can observe from the plot in Figure 4.4, most of the users do not receive any of the Top-20 movies as recommendations. There are some instances of the ”Truncated”

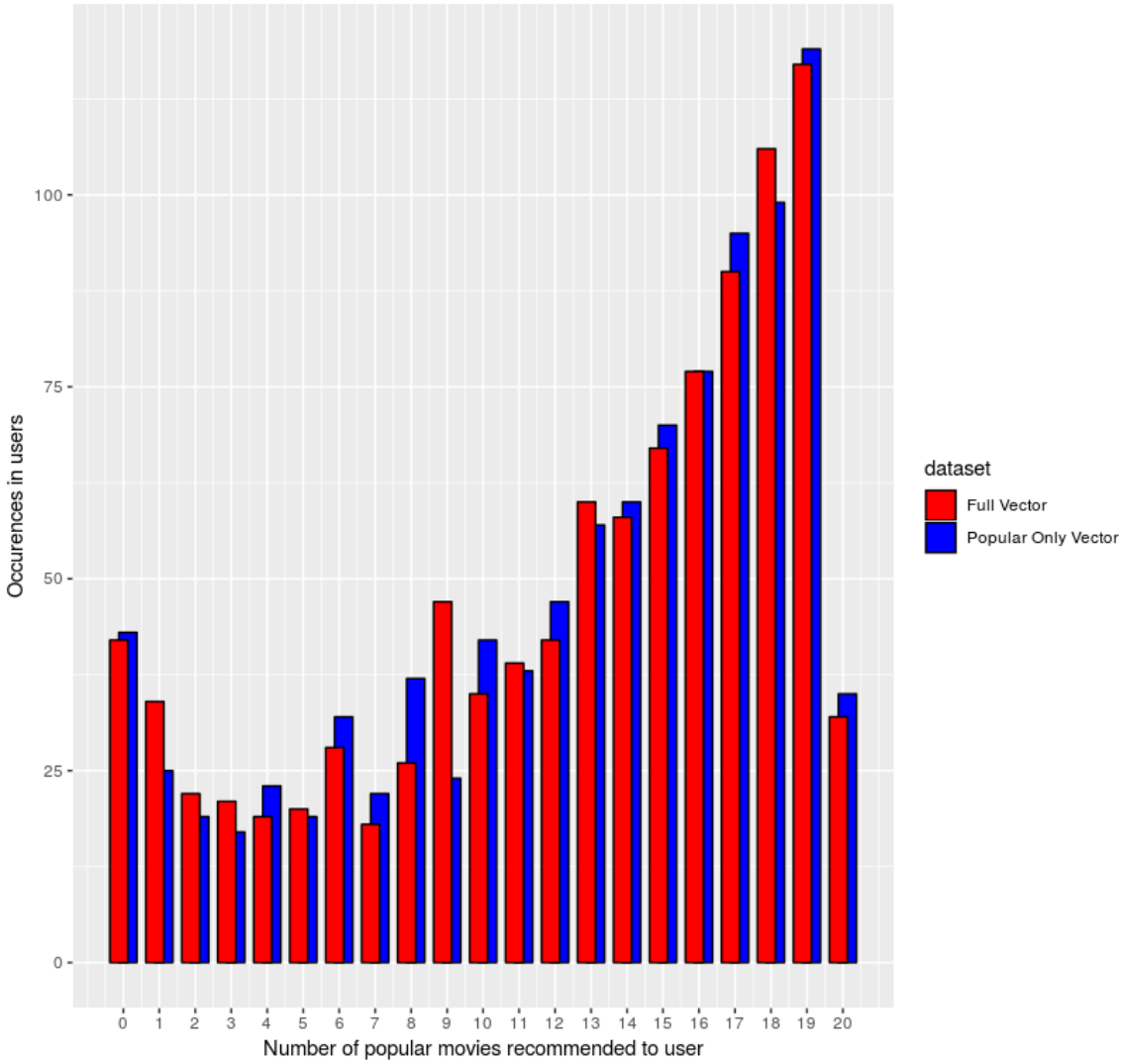

Figure 4.3: Histogram of Number of Popular Movies Recommended to Full User Profile vs Popular Only User Profile(n=1000)

profile receiving only more Top-20 movie recommendations, but there are also cases where the ”Full” profile receives more Top-20 movie recommendations. From this plot, we can conclude that ”truncating” a user's profile does not make there recommendations more or less personalized.

Apart from looking at the overlap with the Top 20 most popular movies in the system, we

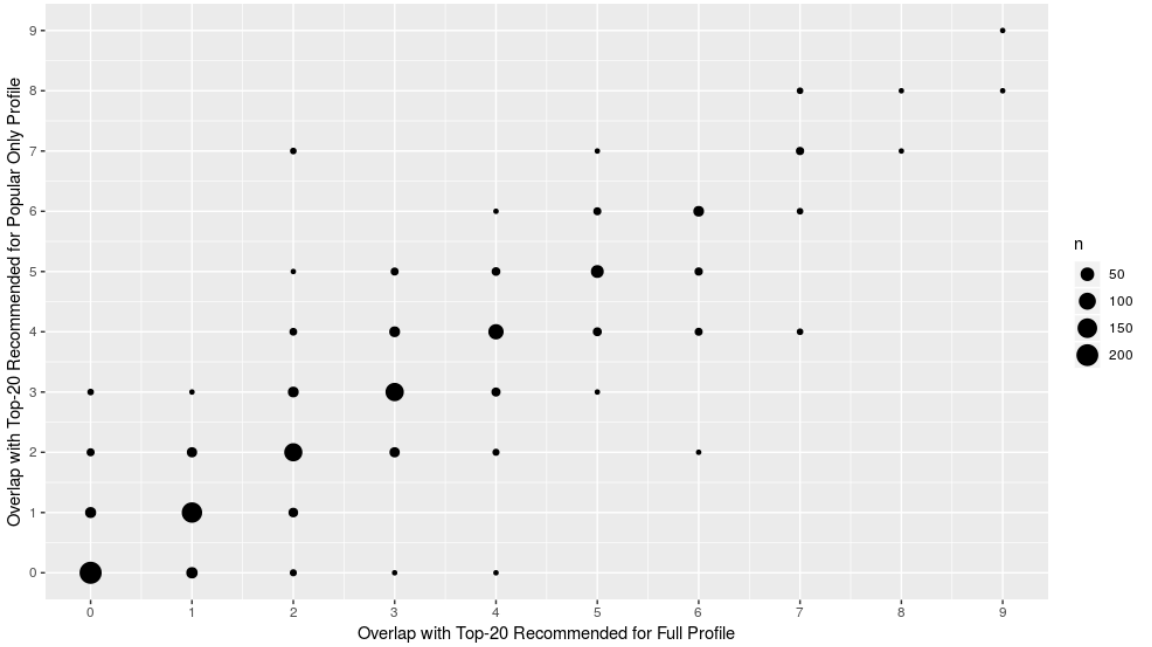

Figure 4.4: Plot of Number of Top-20 Popular Movies found in Recommendations to Full User Profile vs Popular Only User Profile(Total number of users=1000)

also look at the Diversity of a user's recommendation list. The Diversity of a recommendation list is given by the average pairwise cosine distance of all items in the recommendation list. A recommendation list with a moderate amount of diversity is indicative of good personalization. A low diversity score means that the recommendations are very similar to one another. The cosine distance between two vectors u and v is given by :

$$1 - \frac{u * v}{||u||_2 * ||v||_2}$$

which can be considered as " 1 - cosine similarity between u and v". We calculate the pairwise cosine distance of all the item feature vectors in a user's recommendation list for both the "Full" and "Truncated" Profile. As a baseline, we look at the average Diversity of all the movies recommended to either profile. The average baseline diversity to the movies recommended to the "Full" profile is 0.6110. The average baseline diversity to the

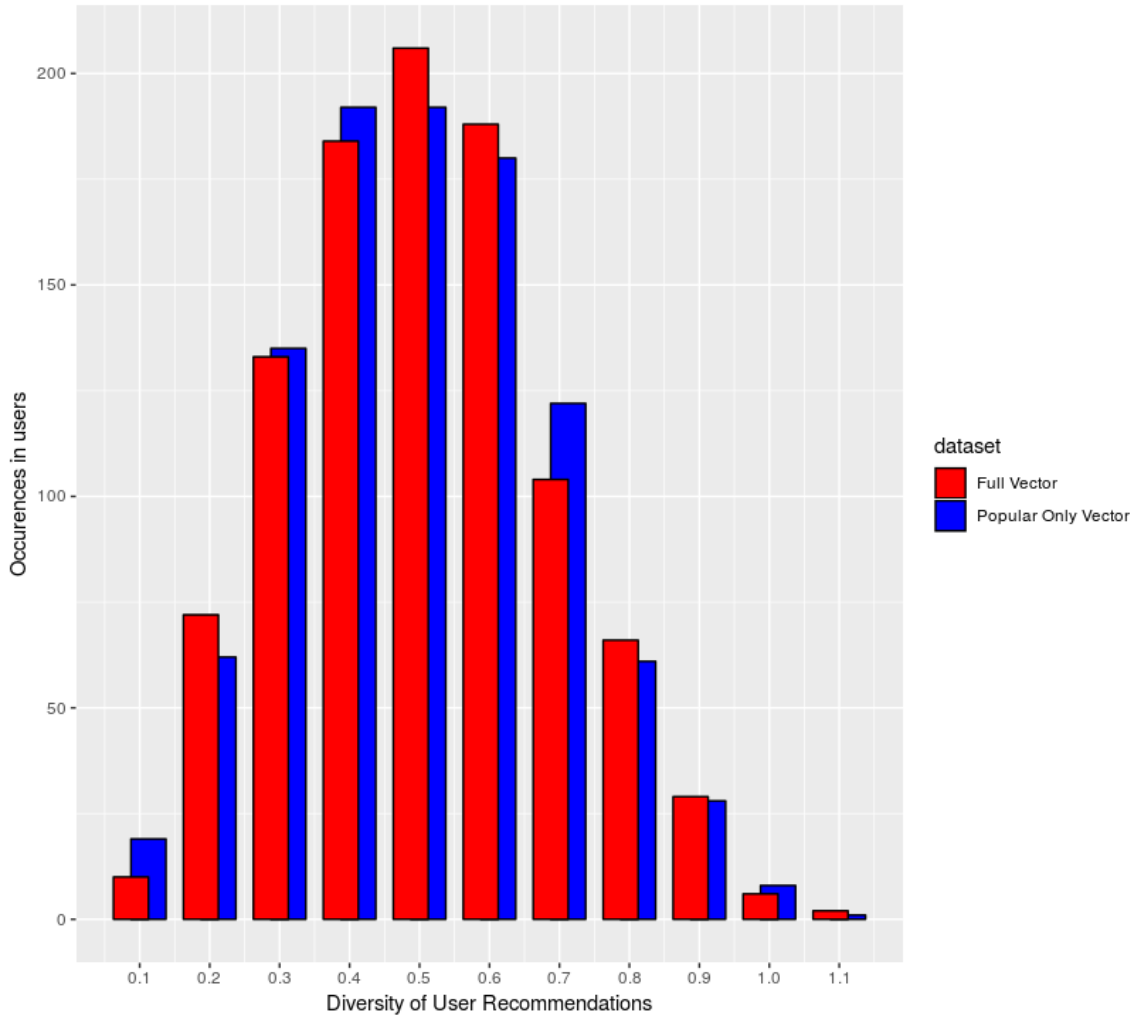

Figure 4.5: Comparative Histogram of Diversity in Recommendation List between Full User Profile and Popular Only User Profile(n=1000)

movies recommended to the ”Truncated” profile is 0.6040. We plot the Diversity scores of the recommendations list of our test users(both the ”Full” and ”Truncated” profiles). From the histogram in Figure 4.5, we can observe that on average the diversity of a user's recommendation is not affected by ”Truncating” their profile. Both of these distribution have an average of 0.5 to 0.6 which tells us that the recommendation lists have the same diversity of recommendations as compared to the baseline we calculated for all movies that are

recommended.

Q2c. How much overlap is there between the top-n recommended items for full and truncated user profiles?

We look at the recommendations generated for two profiles(”Full” and ”Truncated”) for each user. We look at the how many of those recommendations are the same. We measure this as the overlap between the recommendations received by the ”Full” profile and the recommendations received by the ”Popular Only” profile. We plot a histogram of the values of overlap observed between 0 to 20.

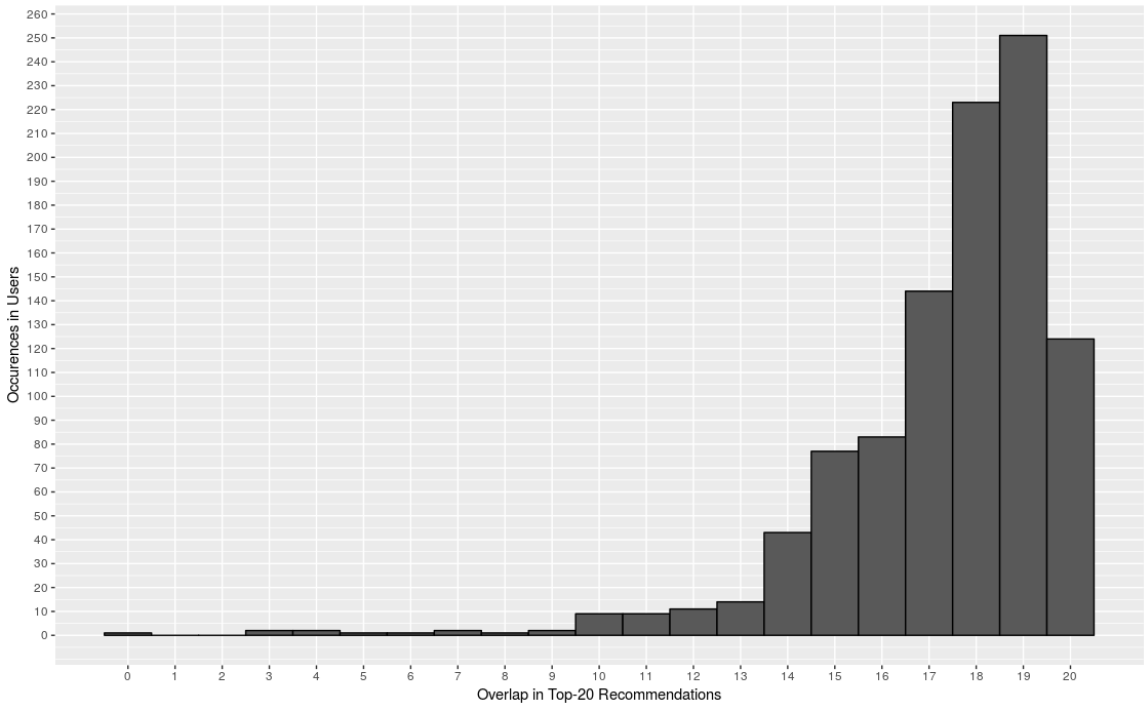

Figure 4.6: Histogram of Overlap Between Recommendations given to Full User Profile vs Popular Only Profile

From the histogram in Figure 4.6 we can see that most users have at least 15 movies that overlap between the two sets of generated recommendations. We can conclude from this histogram that on average, the recommendations received by a ”Full” profile and a

”Popular Only” profile is very high. Thus it is fair to say that a user receives almost the same recommendations even after ”Truncating” their profile.

Q3. What is the effect of removing these ”unpopular” items from a user’s profile on the top-n recommendation metrics?

To analyze the effects of removing ”unpopular” items from a user’s profile, we generate train/test splits wherein we restrict what we can consider for our test set. In our example, we consider the Top-n recommendations of nDCG@20, Precision@20 and Recall@20. We prepare train/test splits from the ratings data of our test users. We consider the ratings data of both the ”Full” and ”Truncated” version of the test users. In each case, we hold out 20 % of a user’s ratings as ratings to test our recommender model on. The remaining ratings of a user(80%) is treated as a new user that we fold into our existing model(we train for the ratings on the already existing item feature vectors). The user feature vectors that are created for these folded-in ratings are recommended for by the recommendation model. The resulting recommendations, along with the held-out test items are compared and metrics are generated on the basis of these comparisons. This evaluation is run multiple times and the average results are taken to ensure that we avoid any issues of bias in sampling. In our case, the entire procedure is run 10 times and the average value of the metrics over these 10 runs are reported.

The top-n recommendation metrics are measures used to gauge the performance of a recommender. nDCG@20 is applying the normalized Discounted Cumulative Gain metric on the list of Top 20 recommendations received for a user. This metric is calculated by calculated the Discounted Cumulative Gain for a given recommendation list divided by the ideal Discounted Cumulative Gain expected(the score that would be generated if the

recommended recommended just relevant items). The formula for DCG@20 is as follows:

$$nDCG_20 = \sum_{i=1}^{20} \frac{rel_i}{log_2(i+1)}$$

where rel_igives us the relevance of the item being recommended at i. The Ideal nDCG also calculates this score if all the items that get recommended are relevant items. For the purposes of this experiment, we need to convert the test ratings that we have to relevance scores. We first normalize the held-out ratings by removing user, item and global biases(obtained from the original 20 million ratings data-set). We then replace all negative ratings with 0's to reflect irrelevant items and also positive ratings with 1's to reflect relevant items. We then compare the Top-20 recommendations that we generate with the held-out test items.

Apart from nDCG, we also have 2 other Top-N recommendation metrics, Precision and Recall, which are also metrics borrowed from the field of information retrieval. Precision is the measure of how many items recommended by the model are relevant items and is given as the ratio of the number of relevant items recommended by the total number of recommendations. Precision is a measure of how precise the recommender is in recommending the relevant items. Recall, on the other hand,is a measure of how many of the relevant items are recommended is the ratio of relevant items recommended divided by the total set of relevant items.Recall is a measure of how many of the relevant items the recommender can recall from the total set of relevant items.In the case of this evaluation, the number of recommended items are fixed at 20. Precision and Recall, at first glance may appear to be very similar but for the same recommended list can have varying values.It is possible that a recommender may recommend a high number of relevant items, but if there are a large number of relevant items for instance, the Recall score would be low. Thus, Precision and Recall can be used to in tandem to truly understand the recommender's ability to recommend relevant items. The mathematical formulae of Precision and Recall are given

below:

$$Precision = \frac{|RelevantItems \cap RecommendedItems|}{|RecommendedItems|}$$

$$Recall = \frac{|RelevantItems \cap RecommendedItems|}{|RelevantItems|}$$

From the results in Table4.1 we can that for most metrics, the ”Popular-Only” profile of the test user on average, scores about the same as the ”Full” profile. This tells us that there isn't a negative affect on top-n metrics on ”truncating” the profile of ”casual”users.

Profile	nDCG@20		Precision@20		Recall@20	
	Pop	Unpop	Pop	Unpop	Pop	Unpop
Full	0.00099	0.00265	0.00080	0.001207	0.00090	0.00692
Popular-Only	0.00135	0.00317	0.00085	0.001288	0.00099	0.00732

Table 4.1: nDCG,Precision and Recall of Casual Users Test Sets

Q4. How does truncating a user's profile affect the accuracy of item predictions?

The prediction accuracy of the recommender model is measured by many measures. For this evaluation, we look at the Root Mean Square Error(RMSE). RMSE is a measure where the error over each predicted is calculated, the error is squared and the mean of these values is calculated and then the square root of this value is taken. To analyze the difference

Test Set	RMSE	
	Full	Popular Only
1	0.8409	0.8243
2	0.8398	0.8374
3	0.8430	0.8456
4	0.8350	0.8341
5	0.8441	0.8336
Mean	0.8405	0.8350

Table 4.2: 5 Crossfold RMSE Test Results

in prediction accuracy in recommender systems, we calculate the RMSE for the ratings of the full profile and popular only profile. We do this by partitioning the two sets of ratings into 5 groups where we create a profile consisting of the ratings of 4 parts and predict for the remaining 1 part. This is repeated 5 times, changing which 4 parts of the the set is considered to train.

From the results in Table 4.2, we see that RMSE scores for ”Popular Only” profile is almost the same as the scores for ”Full” profile.We can be sure in saying that ”truncating” the ”casual” user profile does not affect the ability of the model to predict the user's ratings.

Profile	RMSE	
	Pop	Unpop
Full	0.92068	0.8717
Popular-Only	0.92068	0.8836

Table 4.3: RMSE Scores of Casual Users Test Sets

Q4a. Is the change in accuracy different for predictions on popular vs. unpopular items?

To analyze the difference in metric results between holding out ”popular” items and ”unpopular” items, we begin with the setup that we used for our top-n metric calculation. For that method, we considered a user's ”Full” and ”Truncated” profile and held out the 20 %. In this evaluation, we specifically hold-out 20 % of a user's rated movies that are ”popular”(movies that belong to the Top 2500) and test for this held-out set. We also specifically hold-out 20 % of a user's rated movies that are ”unpopular”(movies outside the Top 2500) and test for this held-out set. We thus have 4 profiles that we fold-in and generate recommendations on. In the case of holding-out ”unpopular movies, we don't hold out any of the ratings from the ”Truncated” test user profiles since these ratings don't contain any ”unpopular” movie ratings.

Table 4.3 holds the results of the results of this analysis. We observe that the RMSE scores are the same for both profiles when we are testing on ”popular” items. However, we the RMSE scores are slightly better for the ”Full” profile on ”unpopular” movies test set. This is expected since the ”Popular Only” profile doesn't contain any information about the user's preference for ”unpopular” movie ratings and so the prediction is slightly worse.

Model	RMSE	
	Pop	Unpop
Full	0.78955	0.793347
Half	0.87329	0.791214
None	0.86058	0.790105

Table 4.4: RMSE Scores of Serious Users Test Sets

Q5. Does truncating ”casual” users profiles to ratings of only ”popular” items affect the performance of the model for ”serious” users?

To analyze the effects of truncating ”casual” user profiles on other ”serious” users involves modifying the train and test data set we are working with. We design three possible training data sets. The first one, ”Full”, involves removing the ”unpopular” movie ratings of all ”casual” users. This amounts to removing approximately 600,000 ratings with a new training set of 19.3 million ratings. The second model,” Half”, involves removing the ”unpopular” movie ratings of half of the ”casual” users chosen at random.This amounts to removing approximately 300,000 ratings with a new training set of 19.6 million ratings. The third model,”None” consists of removing none of the ”unpopular” movie ratings of the ”casual” users. We define our ”test” set by considering a sample of a 1000 ”serious” users(who have rated more than 200 movies). To be consistent with our earlier results, we also generate two held-out ratings sets the three models. We thus have 6 training sets and 2 test sets. We hold out 20% of ”popular” movies for one split and 20% of ”unpopular”

movies for the second test set.Our 6 training sets contain the same ratings as the original ratings for the models except for the held-out data. To calculate the metrics, we train the three models on their respective training sets and generate recommendations(Top-20) and generate predictions for the held-out test data with each model. We then calculate the 4 metrics, nDCG@20, Precision@20,Recall@20 and RMSE using our recommendations, predictions and test data similar to previous questions.

Model	nDCG@20		Precision@20		Recall@20	
	Pop	Unpop	Pop	Unpop	Pop	Unpop
Full	0.0002815	0.000310	0.0007	0.000678	0.000199	0.000662
Half	0.0002853	0.000273	0.0004	0.00024	0.000199	0.001008
None	0.0002908	0.000266	0.0009	0.00051	0.000179	0.000514

Table 4.5: Top-N Recommendation Metrics on Serious Users Test Sets

The results of these tests are in Tables 4.4 and 4.5. From Table 4.5 we can see that the ”Full” model performs slightly worse on held-back ”popular” movie data on nDCG@20 and Precision@20 than compared to the ”None” model. The ”Full” model however, performs better than the ”Half” model on these metrics, however, the difference is slight. The ”None” model performs best for these two metrics. However, ”Full” and ”Half” perform better than ”None” on Recall@20. In the case of the ”Unpopular” movie test set, ”Full” performs better than ”Half” and ”None” on nDCG@20 and Precision@20. However, ”Half” performs better than the other two models on Recall@20. The RMSE scores in Table 4.4 tell us that the ”Full” model generates a better RMSE score for ”serious” users on the ”popular” held out test than the other two models whereas ”None” performs better than ”Full” and ”Half” when holding out ”unpopular” movies.

Chapter 5

Conclusions

We set out this offline evaluation to examine the effects of removing ”unpopular” movie ratings from ”casual” users of a recommender system. We looked at the effects this ”truncation” has on the user's predicition accuracy(”RMSE”) on both popular and less-popular items. We looked at how this truncation might affect the top-n recommendations the model generates for the user by looking at how many popular items the user was recommended, what the overlap was in terms of Top-20 recommendations, and if there was any loss of personalization of the recommendations generated for the user. We also examined the difference in performance in terms of top-n recommendation metrics such as nDCG, Precision and Recall, specifically for both popular and unpopular items. We even examined the effects of this truncation on other more invested users in the system by looking at prediction accuracy and top-n recommendations such as nDCG,Precision and Recall on both popular and less-popular items.

From our results, we can conclude that there isn't a overall loss in information for the users whose profiles are being truncated , although this loss isn't non-zero when it comes to predictions on less-popular items. For the more invested-users in the system, the results aren't as clear, as there is a slight loss of performance when testing on held-out popular items, but there is an increase in scores when testing on held-out less-popular items whereas this is flipped when considering the predictive accuracy for these more serious users.

Overall, we are encouraged by the results of these evaluations as we feel they are a sig-

nificant step towards building a recommender system that would have a simplified interface and a ”lite” version of it’s model for more casual users by truncating their profiles and a ”pro” version for more serious and invested users. We do believe that we need to test this model on real users to measure the effects of this truncation on user satisfaction and behaviour. We believe this is the first example, to the best of our knowledge, of a data-driven approach towards simplifying the recommender system for more casual users.

Bibliography

[1] Robert M. Bell and Yehuda Koren. 2007. Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights. In *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining (ICDM '07)*. IEEE Computer Society, Washington, DC, USA, 43–52. https://doi.org/10.1109/ICDM.2007.90

[2] Dirk Bollen, Bart P. Knijnenburg, Martijn C. Willemsen, and Mark Graus. 2010. Understanding Choice Overload in Recommender Systems. In *Proceedings of the Fourth ACM Conference on Recommender Systems (RecSys '10)*. ACM, New York, NY, USA, 63–70. https://doi.org/10.1145/1864708.1864724

[3] DEREK BRIDGE, MEHMET H. GÖKER, LORRAINE McGINTY, and BARRY SMYTH. 2005. Case-based recommender systems. *The Knowledge Engineering Review* 20, 3 (2005), 315–320. https://doi.org/10.1017/S0269888906000567

[4] R. D. Burke, K. J. Hammond, and B. C. Yound. 1997. The FindMe approach to assisted browsing. *IEEE Expert* 12, 4 (July 1997), 32–40. https://doi.org/10.1109/64.608186

[5] Dan Cosley, Shyong K Lam, Istvan Albert, Joseph A Konstan, and John Riedl. 2003. Is seeing believing?: how recommender system interfaces affect users' opinions. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 585–592.

[6] Michael D. Ekstrand. 2018. The LKPY Package for Recommender Systems Experiments: Next-Generation Tools and Lessons Learned from the LensKit Project. *CoRR* abs/1809.03125 (2018). arXiv:1809.03125 http://arxiv.org/abs/1809.03125

[7] Simon Funk. 2006. Netflix Update, Try This at Home. https://sifter.org/simon/journal/20061211.html

[8] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. 1992. Using Collaborative Filtering to Weave an Information Tapestry. *Commun. ACM* 35, 12 (Dec. 1992), 61–70. https://doi.org/10.1145/138859.138867

[9] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. 2001. Eigentaste: A Constant Time Collaborative Filtering Algorithm. *Information Retrieval* 4, 2 (01 Jul 2001), 133–151. https://doi.org/10.1023/A:1011419012209

[10] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4, Article 19 (Dec. 2015), 19 pages. https://doi.org/10.1145/2827872

[11] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. 1999. An Algorithmic Framework for Performing Collaborative Filtering. In *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '99)*. ACM, New York, NY, USA, 230–237. https://doi.org/10.1145/312624.312682

[12] Nicolas Jones and Pearl Pu. 2007. User Technology Adoption Issues in Recommender Systems. *Proceedings of the 2007 Networking and Electronic Commerce Research Conference* (2007), 379–394. http://infoscience.epfl.ch/record/128435

[13] Yehuda Koren and Robert Bell. 2011. *Advances in Collaborative Filtering*. Springer US, Boston, MA, 145–186. https://doi.org/10.1007/978-0-387-85820-3_5

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

[15] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. 2011. *Content-based Recommender Systems: State of the Art and Trends*. Springer US, Boston, MA, 73–105. https://doi.org/10.1007/978-0-387-85820-3_3

[16] Al Mamunur Rashid, Istvan Albert, Dan Cosley, Shyong K. Lam, Sean M. McNee, Joseph A. Konstan, and John Riedl. 2002. Getting to Know You: Learning New User Preferences in Recommender Systems. In *Proceedings of the 7th International Conference on Intelligent User Interfaces (IUI '02)*. ACM, New York, NY, USA, 127–134. https://doi.org/10.1145/502716.502737

[17] James Reilly, Jiyong Zhang, Lorraine Mcginty, Pearl Pu, and Barry Smyth. 2007. A comparison of two compound critiquing systems. (01 2007). https://doi.org/10.1145/1216295.1216356

[18] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender systems handbook*. Springer, 1–35.

[19] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2000. *Application of dimensionality reduction in recommender system-a case study*. Technical Report. Minnesota Univ Minneapolis Dept of Computer Science.

[20] Badrul Munir Sarwar, George Karypis, Joseph A Konstan, John Riedl, et al. [n. d.]. Item-based collaborative filtering recommendation algorithms. ([n. d.]).

[21] Hideo Shimazu. 2001. ExpertClerk: Navigating Shoppers' Buying Process with the Combination of Asking and Proposing. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2 (IJCAI'01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1443–1448. `http://dl.acm.org/citation.cfm?id=1642194.1642287`

[22] Martijn C. Willemsen, Bart P. Knijnenburg, Mark P. Graus, Linda C.M. Velter-Bremmers, and Kai Fu. 2011. Using latent features diversification to reduce choice difficulty in recommendation lists. *CEUR Workshop Proceedings* 811 (1 12 2011), 14–20.

[23] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. 2008. Large-Scale Parallel Collaborative Filtering for the Netflix Prize. In *Algorithmic Aspects in Information and Management*, Rudolf Fleischer and Jinhui Xu (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 337–348.