Comparative Analysis of Techniques for Data Minimization for Recommender System algorithms

Master's Thesis

Manoj Krishnaraj

Comparative Analysis of Techniques for Data Minimization for Recommender System algorithms

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE
Data Science

by

Manoj Krishnaraj
born in Coimbatore, India

Multimedia Computing Group
Faculty of Electrical Engineering, Mathematics and Computer Science,
Delft University of Technology
Delft, Netherlands
http://mmc.tudelft.nl/

Comparative Analysis of Techniques for Data Minimization for Recommender System algorithms

Author: Manoj Krishnaraj
Student id: 4485742
Email: M.Krishnaraj@student.tudelft.nl

Abstract

Recommender systems (RS) often use a large amount of data for a marginal gain in performance. This thesis investigates the data minimization in Recommender Systems, which is not well studied in the literature. This thesis extends the data minimization principles advocated in GDPR and studies its effects on recommender systems. Minimizing data not only reduces storage and transmission requirements but also has the potential to improve privacy and increase training and prediction speeds. This thesis investigates the effects of reducing the amount of data used to model a recommender system. It evaluates the accuracy of the Biased Matrix Factorization (BMF) algorithm by varying the training data on the MovieLens 10 million ratings (ML-10M) dataset.

In this thesis, four data minimization techniques were used. We reproduced one pervious work and proposed three new data minimization techniques. In the first technique, we confirmed previous work concerning training data analysis, where the data outside the selected temporal window were dropped. The second data minimization technique, user profile truncation, retained the recent N ratings for each of the users while truncating the historical ratings. The third technique improved the user profile truncation by selectively truncating a percentage of user's historical ratings. In the fourth technique, a long user profile was split into smaller pseudo-user profiles.

Analysis of the results is conducted. The most interesting results come from the third data minimization technique. Here, we show that truncating a percentage of the least recently active long user-profiles does not damage the performance and may slightly help. 60% of the long users can truncate their profiles to 20 ratings with minimal impact on the performance. Based on the results, we conclude that a substantial amount of data can be dropped without a large impact on the performance. The results hold for ML-10M dataset. It should hold for other datasets. The privacy implications of data minimization warrant future work. The proposed techniques serve as a guide for future research in data minimization of recommender systems.

Keywords: Recommender System, Collaborative Filter, Data Reduction

Thesis Committee:

Chair:	Prof. dr. M.A. Larson, EEMCS, Delft University of Technology *(supervisor)*
Committee Member:	Dr. C. Hauff, EEMCS, Delft University of Technology
Committee Member:	Dr. E. Isufi , EEMCS, Delft University of Technology

Acknowledgements

I want to express my deepest gratitude to Prof. dr. Martha Larson for believing in me and supporting me throughout the journey of this thesis. I extend my gratitude to Manel Slokom for her valuable feedback and guidance.

I would like to thank Dr. Claudia Hauff and Dr. Elvin Isufi for being part of my thesis committee amid their busy schedule.

I thank Leonie Boortman for her invaluable support. I want to thank friends and the members of the graduation group. Shout out to Vimal and Anamitra, who have encouraged me a lot for completion.

Finally, and most importantly, I thank my parents, who started it all.

Manoj Krishnaraj
Delft, The Netherlands
November 2019.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

BMF	Biased Matrix Factorization
CBRS	Content-based recommender system
CF	Collaborative Filtering
CFRS	Collaborative filtering recommender system
CV	Cross-validation
GDPR	General Data Protection Regulation
MAE	Mean Absolute Error
MF	Matrix Factorization
ML-1M	MovieLens 1 million ratings
ML-6M	MovieLens 6.6 million ratings
ML-10M	MovieLens 10 million ratings
ML-20M	MovieLens 20 million ratings
RMSE	Root Mean Square Error
RS	Recommender System

Chapter 1

Introduction

Recommender Systems (RS) are everywhere: the ads that we see, the news and the media that we consume, the items that we shop, the clothes that we wear, and the MOOCs that we enroll are all recommended to us in one way or other. Though Recommender System (RS) primarily evolved with the internet, its reach extends well beyond the internet. Most users are unaware of the extent to which these systems collect and retain their data.

Though the amount of user-generated data grew exponentially in the 2000s with the advent of social media, processing them was still a problem. But with the emergence of the big data processing techniques in the 2000s, all of the user-generated data could now be processed and retained indefinitely. As a consequence, tech companies greedily started capturing more and more user information than necessary for potential future data processing. The future data growth is predicted by Seagate to grow exponentially.

> "The data-driven world will be always on, always tracking, always monitoring, always listening and always watching - because it will be always learning."
>
> (IDC White Paper[1] — The Digitization of the World)

Many tech giants, including Amazon, Google, Facebook, and Netflix, rely on RS and use it as one of their core technologies. They have unprecedented access to user information. New insights derived from this big data led to the hyper-targeting of the users to increase their profits. With a lack of laws governing the data use, these tech companies designed the data processing pipelines to exploit the user's data. Facebook and Google are recipients of massive fines for mishandling data.

European Union(EU) parliament has enacted General Data Protection Regulation (GDPR) [44] with stringent requirements on data processing and retention, which came into effect from May 2018. One of the main principles behind GDPR is *data minimization*, which limits the data collection. Also, the users have the right to be forgotten, resulting in the obligation of the data processing entity to erase such user's data from all data processing. This thesis is inspired by the data minimization principle and is motivated in the following section.

1.1 Motivations

Data Greed

The research related to RS surged with the million dollar challenge *The Netflix Prize* [6]. The aim of the contest was to improve the performance of an anonymized ratings dataset with 100 million ratings. Currently, most research in the field of RS tends to guzzle more data to achieve infinitesimal increases in overall performance. It has become the norm to

[1] https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf

retain and use every bit of user information. This data greed exposes the users in the dataset. Parts of the Netflix dataset was de-anonymized leading to the withdrawal of the dataset. In spite of the privacy concerns raised by [35, 17, 18], this dataset is still widely used in academia and research.

> "Once released, information is hard to control. Thus, over time, the more information and data can be linked and analyzed, the higher the likelihood of being able to make sensitive inferences from it for larger groups of people."
>
> (Alessandro Acquisti — Carnegie Mellon University)

By retaining more data, the users are more susceptible to attacks. But, with newer policies like the right to be forgotten, the user can remove some or all of their data from processing. Due to the recency of this law, the majority of the research in the literature does not consider the above restrictions. Also, as a result of the knock-on effect, the trained models have to be retrained without the user's data, which has the potential to disrupt algorithms and trained models. The effects of such removal are not studied extensively at the moment [47, 46]. One of the goals of this thesis is to study the effects of limiting the data retained per user.

Data minimization

Data minimization limits the data that can be collected from individuals to include the only relevant or necessary information that is essential to accomplish a specific purpose. The GDPR law have enforced privacy and data minimization in many fields, including *Artificial Intelligence(AI)*. GDPR creates an exciting challenge as the AI systems must now employ the principle of data minimization as part of their design before it can learn from the user data. One of the goals of this thesis is to extend the data minimization principle to RS and systematically determine the extent to which the amount of data used in the training of the recommender system could be reduced.

The above goals are formally defined as research questions in the following section.

1.2 Problem statement

The primary motivation for this thesis is to determine the impact of reducing the amount of data used for training a recommender system. The main objective of this thesis can be summarized in the following research question:

> **How can we reduce the amount of data used to train a recommender system?**

Data reduction is ongoing research in data mining. According to [24], there are two ways to achieve data reduction. The first one is to reduce the number of samples. The second way is to reduce the number of features. This thesis focuses on the former problem, reducing the number of samples. Re-sampling is a statistical method to select samples and is widely used to reduce the number of samples. Unlike re-sampling, data minimization refers to the practice of limiting the collection of user data that is necessary to accomplish a specified purpose. Henceforth, we use the term data minimization to refer to the principle of reducing the amount of data used for modeling the RS.

Data minimization in RS can be achieved by training data requirement analysis [31], which is similar to differential data analysis [13]. The training data that is used to model the RS is varied to assess the corresponding differences in performance. In the context of recommender systems, [31] uses temporal windows to reduce the amount of data needed for training. [47] proposed time-based user-controlled filters where ratings between temporal windows are selected, and users retain N_{days} of ratings. Group recommendations [7, 26],

on the other hand, reduce the number of users in the system by grouping similar user profiles into virtual users. The above techniques are closely related to the proposed data minimization techniques in this thesis.

1.3 Research objectives

To address the main problem statement defined in Section 1.2, we introduce the four research questions as data minimization techniques. While the first technique reproduces an earlier work, the remaining three techniques are novel. The four research questions, along with their objectives, are briefed in this section.

- ***RQ1:*** **How does the temporal window truncation of training data affect the performance of the recommender system?**
  Objective 1: We aim to get a better understanding of the changes that happen when we reduce the size of the training data in terms of temporal windows. We compare the performance by varying the size of the temporal windows. The methodology is similar to earlier work done by Larson et al. [31]. We pursue this research question in Chapter 4.

  A *temporal window* contains a fixed number of user ratings that are extracted between two points from a temporally ordered dataset.

- ***RQ2:*** **How does the truncation of users' historical rating profiles affect the performance of a recommender system?**
  Objective 2: We aim to understand the impact of reducing the size of the user's profiles by keeping only a certain amount of the latest ratings and truncating the rest for all the users. We term this data minimization technique as user profile truncation. We pursue this research question in Chapter 5.

- ***RQ3:*** **How many users can truncate their profiles without affecting the overall performance of the system (on average over all test users)?**
  Objective 3: We aim to get a better understanding of the user profile truncation from the previous research question to a select percentage of users in the system. By selective applying the user profile truncation, we aim to reduce the negative impact of We pursue this research question in Chapter 6.

- ***RQ4:*** **If long user profiles are split instead of truncated, does this help the overall performance of the system (on average over all test users)?**
  Objective 4: We want to find whether we can achieve data minimization by dividing the profiles of long users into pseudo-users with shorter profiles. Unlike the previous research questions, we do not remove any user ratings. We pursue this research question in Chapter 7.

1.4 Contributions

The contributions of this thesis are:

- We review the literature related to data minimization in RS.
- We confirm the results of previous work on temporal window truncation in [31] and provide additional insight into the effects of the presence of cold-start users.
- We analyze the impact of truncating all of the user's profile history and point out that the truncation on long users does not substantially decrease the accuracy.

- We propose an approach for selective user-profile truncation, which demonstrates the usefulness of data minimization. We emphasize the need for long profiles for better recommendations.
- We examine two approaches for splitting long user-profiles into smaller pseudo-user profiles. We show that reducing the data of long users by splitting does not impact the overall performance.

1.5 Document structure

This thesis document is divided into eight chapters. Chapter 2 provides a background on RS and contains a review of the relevant literature. Chapter 3 describes the conceptual framework used in this thesis. Chapter 4 addresses the first data minimization technique, temporal window truncation. Chapter 5 proposes the second technique, user-profile truncation, and is improved by the selective user profile truncation in Chapter 6. Then, Chapter 7 examines the impact of splitting long user profiles. Finally, Chapter 8 outlines the main conclusions of this thesis and identifies recommendations for further research.

Chapter 2

Background and Related work

This chapter provides the background on recommender systems, types of recommender systems in Section 2.1, and Section 2.2 respectively. A brief introduction to collaborative filtering algorithms is presented in Section 2.3. Previous work done in the literature relating to this thesis are provided in Section 2.4.

2.1 Recommender system

The amount of user-accessible information has grown exponentially since the dawn of the internet, and this trend is bound to continue resulting in information overload. As a consequence, users are facing increased difficulty in finding and retrieving relevant information. Information filtering system helps to manage the information overload by presenting limited and relevant information to the users. A recommender system is a subclass of information retrieval systems that guide the user in a personalized way to retrieve interesting and useful items from a large collection. The recommender system relies on user feedback to capture a snapshot of the user's information needs. It then uses user modeling to tailor content to the users. This section provides a brief on the most essential concepts in recommender systems that are relevant to this thesis.

Terms and definitions

The terms relating to recommender systems that will be used throughout this thesis are defined below.

- **User:** The end user who interacts with the recommender system either by providing opinions or requires a recommendation.
- **Item:** The general term used to determine what is recommended to the users.
- **Rating:** Refers to a recorded interaction between a user and the RS and is detailed in Section 2.1. r_{ui} denotes the rating given by a user u for the item i.
- **User profile:** The set of all ratings provided by a particular user.
- **User-Item matrix:** The matrix representation of all the users and items two dimensions. Is is also referred as the ratings matrix denoted by R. The rows represent the users and the columns represent the items. Each entry in the matrix denotes the rating for the corresponding user and item. Figure 2.1 illustrates a simple User-Item matrix.
- **User bias:** User bias captures the user's disposition in rating an item.

	$item_1$	$item_2$	···	$item_n$
$user_1$		5	···	1
$user_2$	1	4	···	
$user_3$	5	2	···	5
...	...	...	...	...
$user_n$		1	···	

Figure 2.1: A simple User-Item matrix using a discrete rating scale with a range of 1 to 5.

Rating

The user's opinion forms the source of input data for the recommender system. The opinions of a user are usually limited as they can rate only a small portion of all the available items. History of the user's opinion is then used to recommend content. The user's opinion is usually captured either as explicit or implicit feedback. The captured user's interaction can be categorized into one of the following scales:

- **Unary:** User can express an only preference or non-preference for an item. Examples: like, favorite, and ignore.
- **Binary:** User can express both preference and non-preference towards an item. Examples: like/dislike, and thumbs up/down
- **Discrete:** User preference towards an item is expressed as a discrete value on an N-point Likert scale, usually in the form of stars.
- **Continuous:** User preference towards an item is expressed on a continuous scale, usually using a slider.

In explicit feedback, the user preference is captured in binary, discrete, or continuous scale. Explicit feedback allows users to input both positive (high score) and negative (low score) feedbacks. One of the most commonly used scales is the five-star rating scale, where users rate from one to five. For example, the MovieLens 10 million dataset [23], which is detailed in Section 3.2, captures explicit user feedback on a scale of one to five in 0.5 increments. Though explicit feedback is reliable, it is hard to collect the user's opinion as many do not prefer to disclose it due to privacy concerns [35].

In the case of implicit feedback, the user's opinion is inferred from the user's behavior. User interactions such as likes, time spent on an item, and clicks are captured as implicit feedback on a unary or a binary scale. Unlike explicit feedback, the implicit feedback can only capture positive feedback as it is challenging to infer the disliked content. For example, a user may interact with an item without actually liking it. The system may still capture it as a positive interaction. Though both forms of feedback can yield noisy data, the implicit feedback is far more prone to noisy data. In some cases, the explicit dataset is transformed into an implicit dataset by using certain threshold conditions.

Recommendation task

Recommendation task refers to the actions taken by the RS in order to suggest new content to the user. The RS uses the history of user preferences as input and outputs personalized recommendations. [21] categorizes the recommendation task into three major groups as:

- **Prediction task:** The system predicts the user's rating for an item over a set of items.

- **Recommend task:** The system recommends an item or a list of items that the user is likely to prefer. A truncated list of top-N items is preferred. Ranking algorithms are used to select the top-N items.
- **Utility maximization task:** The value of the items is determined by a utility function. The system recommends the best items to the user by maximizing the overall utility value of the recommended items.

In this thesis, the recommendation algorithms that perform the prediction task and the recommend task are selected. The type of the recommendation task determines both the evaluation metrics and the datasets used for experimentation as seen in sections 3.4, and 3.2 respectively.

2.2 Types of recommender systems

The literature on recommender system [37] broadly classifies the different techniques into three: content-based recommender system, collaborative filtering recommender system, and hybrid recommender system. This section provides a brief on the types of recommender systems. Collaborative filtering is detailed further in Section 2.2.1.

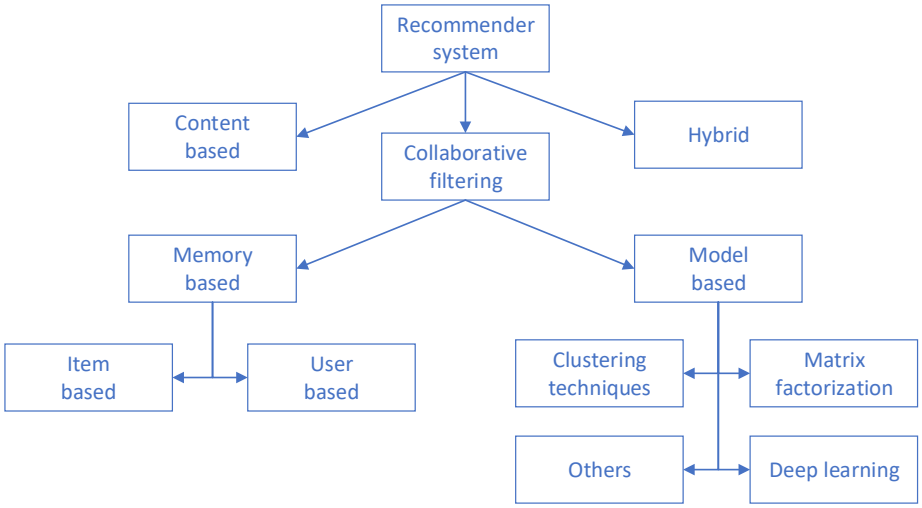


Figure 2.2: Classification of recommender system

2.2.1 Collaborative filtering recommender system

Collaborative filtering recommender systems provide suggestions based on the preferences of other like-minded users [37, 17]. It recommends items that other users with similar tastes have liked in the past based on the similarity in the rating history of the users. The user-item interactions are usually represented in the form of a user-item matrix as shown in Figure 2.1. In the UI matrix, the rows represent the users and the columns represent the items. The user-item interaction is captured in the cell corresponding to the specific user and item. The cells can represent the interaction in case of the implicit feedback or the rating in the case of the prediction task. The main objective of collaborative filtering is to determine the missing value in the UI matrix for a give user and item based on the preferences of the other users.

Collaborative Filtering (CF) is one of the most successful recommendation techniques [17]. The number of publications relating to CF in the literature highlights the increase in

use of CF in both academia and industry. Recently, deep learning based algorithms for CF have gained increased popularity [49]. In spite of its wide use in the industry, CF faces limitations. The limitations of CF relating to this thesis are detailed in Section 2.2.1. The literature on RS classify collaborative filtering into two groups: model-based and memory based as seen in the following section.

Memory-based collaborative filtering

Memory-based collaborative filtering algorithms use the similarities of either users or items for recommendations [17]. They are classified into two: user-based and item-based collaborative filtering. User-based models make predictions for a user-item pair based on the aggregated history of ratings for the item by similar users. Item-based models make predictions for an item as a weighted average of similar items. The memory-based recommendations are structured and can provide explanations for the recommendations. It directly uses the stored ratings without the need for training. Though they are easy to implement, they do not scale well for large real-world datasets. The study of memory based CF is outside the scope of this thesis.

Model-based collaborative filtering

Model-based collaborative filtering trains a model from the available ratings in a training dataset. The trained model is then used for making predictions. There are many model-based CF methods. Some of the model-based methods include Decision and Regression Trees, Rule-based CF, Naive Bayes CF, and Latent Factor Models. The most popular model-based CF method is Matrix Factorization (MF) which uses latent factors. Model-based CF is used for experiments in this thesis and is detailed in Section 2.3.

2.2.2 Factors affecting collaborative filtering approaches

Some of the limitations affecting collaborative filtering approaches that are closely related to this thesis are discussed briefly.

- **Sparsity:** In the real world, the number of users is very high resulting in a large UI matrix. The number of items rated by each user is less. The majority of the UI matrix is empty resulting in a very sparse user-item matrix. As a consequence, the performance of the recommender system is reduced. This limitation of CF is called as sparsity problem.
- **Cold-start:** Cold-start problem in CF occurs when the system does not have enough information about new users or items. Both users and items are affected. A new item without any ratings might not be recommended until enough users rate it. Similarly, it is challenging for RS to make suggestions for users without enough ratings. New users without any ratings in training data set are cold-start users. Similarly, items without any ratings in the training set are cold-start items. The presence of cold-start users and items reduces the performance of the RS. Sparsity and cold-start problems are related.

2.2.3 Content-based recommender system

Content-based recommender system (CBRS) rely on user preferences and the item features to recommend items. In this approach [37], the system analyzes a collection of items previously rated by the user. The RS uses the domain knowledge concerning users and items for feature extraction. A profile of the user's interest is built based on the item features rated by the user. Such profiles are created for each user or item. Then recommendations are made by matching up the features of the user's profile against the features of the item.

The strengths and weaknesses of CBRS are different to that of CF [37]. In the CBRS, the user profiles are independent of the other users. Hence it can recommend new items and does not suffer from the user cold-start problem. Additionally, CBRS provides transparency as it can illustrate the reasons for recommending a particular item.

However, it is challenging to design CBRS. Enumerating all the features of an item that could influence the user's preferences is challenging. Features that are good for one domain may not be as effective for another. Hence extensive domain knowledge is a pre-requisite for extracting relevant features. For the CBRS to profile the users, enough number of items must be rated by the user. Lack of ratings results in poor profiles and unreliable recommendations. Also, the CBRS suffers from the serendipity problem and it is usually over-specialized to address a specific use case.

2.2.4 Hybrid recommender system

Hybrid recommender system combines two or more recommender systems, usually a content-based and a collaborative filtering recommender system. [1, 12] outlines the ways in which the hybridization of RS can be achieved. Some of the strengths and weaknesses of CF and CBRS are complementary [37]. The hybrid RS not only minimizes the effects of the limitations of an individual recommender but also benefits from their advantages. Hence hybrid recommender systems have improved prediction accuracy over either one of the above. Cano et al. [12] reviews the state of the art developments in hybrid RS.

2.2.5 Why collaborative filtering?

In general, it is difficult to compare results from different recommender systems. [39] discusses the challenges involved in comparing the results of RS algorithms. The CBRS are domain specific. As was previously stated in Section 2.2.3, the features of the CBRS must be designed specifically for each domain. For this reason, the comparison of results between CBRS is not meaningful. Furthermore, [12] shows that almost all hybrid recommenders use at least one CBRS algorithm. As a consequence, both CBRS and hybrid RS are not included in the scope of the experiments in this thesis. [8, 39] show that comparative studies on CF algorithms are possible under controlled conditions. We limit the scope of this thesis to CF algorithms using controlled evaluation strategies [8, 39].

2.3 Collaborative filtering algorithms

This section details the different collaborative filtering algorithms used in this thesis. Matrix factorization is one of the classical CF algorithms widely used in research and is detailed in Section 2.3.1.

2.3.1 Matrix Factorization

MF is one of the most popular techniques used in collaborative filtering of recommender systems. A latent factor model is used by the majority of MF models. In a latent factor model, characteristics contributing towards a rating are inferred from the rating patterns using a number of latent factors, say k. MF captures the user-item interactions in the latent feature space. Stochastic gradient descent and alternating least squares are two commonly used learning techniques for MF [30]. The MF model is visualized in Figure 2.3.

For a given set of users U, and set of items I with user-item ratings matrix R of size $\mid U \mid \times \mid I \mid$, the matrix factorization with k latent factors is given by:

$$R \approx P \times Q^T = \hat{R} \tag{2.1}$$

In the above equation 2.1, two latent feature matrices P and Q are found such that their product equals to R. The matrix P's dimensions are size $\mid U \mid \times k$. Similarly, the matrix Q

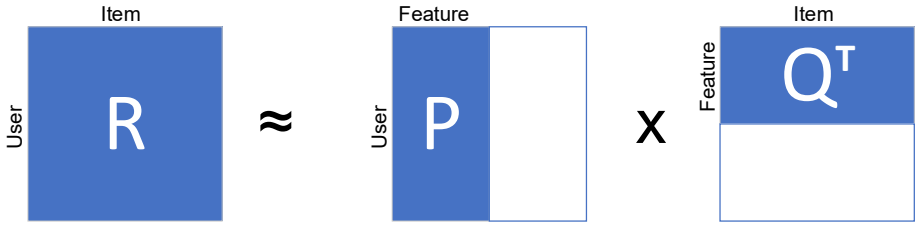


Figure 2.3: Matrix factorization

is of size $| I | \times k$. The matrices P and Q represent the latent features of users and items respectively as illustrated in Figure 2.3.

The user-item interaction is modeled as the dot product of corresponding user and item latent features. Therefore the predicted rating $\hat{r}_{ui}$ for the user u and the item i is given by

$$\hat{r}_{ui} = p_u \cdot q_i^T \tag{2.2}$$

A number of MF models exist in the literature [37, 30]. Some of the widely used models include singular value decomposition, principal component analysis, probabilistic matrix factorization, and BMF. Cunha et al. in [14] use meta-learning to determine the best models for different datasets and recommendation tasks. Based on their conclusion, BMF is one of the best models for rating prediction for a number of datasets. We look at the workings of the BMF in detail in the following section.

2.3.2 Biased Matrix Factorization

The user's disposition towards the rating scale creates biases in the CF data. Two important biases are user bias and item bias. Some users give higher ratings than others resulting in user bias. Some items receive higher ratings than others resulting in item bias. The first order approximation of the above bias, b_{ui}, for a rating r_{ui} is as follows:

$$b_{ui} = \mu + b_i + b_u \tag{2.3}$$

In the above equation, μ is the overall average rating, b_u is the user's deviation from average, and b_i is the item deviation from average. The biases defined in equation 2.3 are added to the basic MF in equation 2.2 resulting in equation2.4.

$$\hat{r}_{ui} = \mu + b_i + b_u + p_u \cdot q_i^T \tag{2.4}$$

BMF [30] incorporates four components: global average, user bias, item bias, and user-item interaction for rating predictions. Learning in BMF is done by stochastic gradient descent as in the basic MF. As BMF uses explicit user and item bias, it predicts ratings more accurately when compared to the basic MF. The effects of biases and other temporal effects are elaborated in [30, 28].

2.4 Related Work

Recommender System (RS) can be applied to solve several open problems promoting the rapid growth of active research in this field. The amount of works in the literature is increasing rapidly. Textbooks [2, 37, 38] outlines much of the work done in the field of RS in the preceding decade. Extensive surveys [1, 43, 49] examines the evolution of the recommender systems from its origins to the state of the art developments. Collaborative Filtering (CF) is widely used in the industry and is one of the most successful recommender

systems in practice. The important survey [17] explains the concepts and advances in CF. Surveys [43, 8, 41] illustrates the various advances in CF at different points of time and discusses open problems. The scope of this thesis is limited to CF algorithms, as discussed in Section 2.2.5.

This thesis proposes a comparative study of the evaluation of training subsets using CF algorithms. Among the works in the literature, [8, 39, 32, 10] do a comparative study in RS. [8, 32] compares several CF algorithms and study their behavior, and limitations. [8] compares the different techniques of evaluating CF algorithms and follows two approaches for selecting the training subset and evaluation data. [39] critiques the different recommender system frameworks and argues for the need of a set guidelines to evaluate, benchmark, and compare RS results effectively. The survey [10] differentiates existing evaluation protocols and proposes a set of guidelines to enable a comparison of results. The methods used in [8, 39, 32, 10] form the basis for the design of the training data analysis experiments in this thesis.

Time-based ordering of the ratings

In this thesis, the time-based ordering of the rating data associates a temporal context to it. The splits for evaluation produces a training set which contains temporally older ratings than the test data. This ordering mirrors the real-world scenario. [48] shows that the performance of the temporally split data is lower than the random division of data. Time-based user profiling was first explored by [33] for RS. Survey [45] provides a comparative study of the different time-aware CF algorithms. The comprehensive survey by Campos et al. [10] identify the areas in which time-aware RS have diverging characteristics and establish new methodologies. They discuss the different characteristics of temporal data, such as recency, drift, decay, and seasons. Koren et al. [28, 29] first integrated temporal dynamics modelling into MF. Survey [3] focuses on temporal effects of MF in detail. This thesis leverages the recency effects of the temporally ordered data.

Training data requirement analysis

Training data requirement analysis proposed by Larson et al. [31] is one way to achieve data minimization. Their work is similar to differential data analysis [13]. The training data that is used to model the RS is varied to assess the corresponding differences in performance. Varying the training data changes the underlying characteristics associated with it. Many papers in the literature [37, 32, 8] highlight the importance of data analysis for producing effective recommendations. [32] studies the dependencies on data size and density. [8] study the influence of user-item matrix density on the type of RS algorithm. The training data selection and the characteristics of training data are analyzed in this thesis.

Some of the user preferences contribute more to determine the accuracy of the recommendation system while others less so. Chow et al. [13] characterize the data that contributes the most to the accuracy of the recommender algorithm as important data. They employ differential data analysis, a technique similar to differential cryptanalysis, where the insight is gleaned from the analysis of slightly different inputs to the system. In this approach, the data is chunked into smaller subsets and is filtered based on its usefulness. The importance of a selected chunk of data is determined by comparing the accuracy in its presence and absence. The data chunks that contribute marginally are pruned. For the location dataset, Gowalla check-ins, the removal of 40% of the training data did not significantly impact accuracy. In the case of the MovieLens dataset [23], the high and low user ratings at either end of the rating scale are of more importance for the rating prediction task than the ratings in the middle of the scale. [13] shows that the selection of data based on the importance not only helps to reduce the amount of data required to maintain the accuracy of the recommender system but also potentially increase privacy.

The research questions in Section 1.3 intend to determine the importance of the temporally ordered user ratings. First, we confirm the earlier work on time window truncation [31]. Then we introduce three new techniques for data minimization. The related literature for each of the research questions is discussed in the following sections.

Time window truncation

The work by [20] is one of the earliest works which relates to the first data minimization technique in this thesis. It evaluates time-filtering of user preferences based on temporal windows and introduces the time-aware varying window experimentation. The first technique follows the prior work done by Larson et al. [31], in which the authors take a position on the responsible recommendation. They propose that a recommender should use the minimum amount of data necessary for training the RS when possible. They split the temporally ordered dataset into 11 segments and construct temporal windows of different sizes to vary the size of the training data. Though their work shows that a smaller amount of training data can result in a similar level of performance, it has several shortcomings. Firstly, the authors in [31] did not account for the effects of varying the ratio of training to test ratio in their work. Though the experiments used a constant test set, training sets vary based on the history lengths. In their experiments, the ratio of the train to the test data size varied from 1:1 to 8:1. One drawback of the above method of selecting the train and the test set is that there may be many users without any ratings in the test set. The authors did not account for the changes in the underlying characteristics of data such as density, number of cold-start users and items, and number of trained items and users. Similarly, the authors also did not take into consideration the user rating distributions through time. In this thesis, we point out the impact of the characteristics of the data due to temporal window truncation in Chapter 4.

User-profile truncation

The second data minimization technique truncates the older ratings in a user-profile while retaining recent ratings. [15] introduced time-based weighing of recent user preferences and studied the effects of drift and decay [16] in RS. [15, 16, 10] establishes that user preferences near the recommendation date have more value compared to older preferences in most cases. [9] demonstrates the recency effects. However, [10] cautions that some older preferences could be essential to address the serendipity problem and suggestion of new items. [10] argues that time-based truncation of user-profiles is an extreme case of time-based weighing. We point out the effects of truncating the user profiles in Chapter 5.

Selective user-profile truncation

Recent work by Wen et al. [47] relates to the third data minimization technique in Chapter 6. The authors analyze the training data requirements of the recommender using time-based user data filtering. They simulate the experiments with controlled erasure of the users' historical ratings. The performance of the algorithms under different population levels is compared against the baseline without any filters. One of the main shortcomings of this paper is that the authors do not consider the characteristics of user-profiles prior to selection for filtering. Randomly selecting a percentage of users for filtering raises questions concerning the reproducibility and the validity of their results. Another shortcoming is their arbitrary selection of short segments of the MovieLens dataset without considering the characteristics of the data. Moreover, the author's choice of filtering N_{days} of data may increase the number of cold-start users over time as there exist a large number of inactive users in the system who sporadically rate as illustrated in Figure 3.8.

An extreme case of such controlled filtering is the concept of data strikes proposed by Vincent et al. [46]. They define "data strike" as the scenario in which users take collective action against the tech giants like Google by boycotting the use of their business-critical

systems, including recommenders systems. [46] simulate data strikes by starving the recommender models of training data and measure its impact on the MovieLens datasets (ML-1M and ML-20M). They conduct focused campaigns wherein a group of users withholds all of their data from training. A small amount of withheld data was sufficient for a decrease in the performance in the case of the ML-1M dataset. However, as the dataset grows large, the effectiveness of the data strikes dwindles. It is similar to Wen et al.'s user-controlled data filtering [47]. Unlike [46], where the full user profile is withheld, the experiments in [47] withhold only a part of user preferences. The authors of [46] consider [47] as a partial data strike. Similarly, the second and third data minimization techniques could be considered as a form of partial data strike.

Long user-profile splits

The fourth data minimization technique in Chapter 7 reduces the number of ratings in a user by splitting the users. This technique is related to user splitting in [5], where user profiles are split into micro profiles based on the context of time. Converse to this approach is group recommenders. Unlike group recommendations, where several users are grouped, we split long users, thereby reducing the number of ratings per long user dropping any ratings. [27] partitions the users' profile based on three strategies: random, genre-based, and clustering. The survey [7] focuses on different types of group recommenders, among which, virtual user approach is of interest to us [26]. A virtual user is an artificially created user profile for similar users in a group [26]. [40] proposes a recommendation framework based on groups as a method to preserve users' privacy.

Chapter 3

Experimental Design

In this chapter, we define a general methodological framework which forms the basis for the experiments discussed in the following chapters. Firstly, we give an overview of the main steps to perform the offline evaluation of recommender systems in Section 3.1. Next, the datasets used in the experiments and analysis of the datasets are presented in Section 3.2 and Section 3.3 respectively. The different evaluation metrics used to measure the performance of the recommender system is detailed in Section 3.4. Finally, the recommender framework along with the implementation details and tools used for experimentation marks the end of this chapter in Section 3.5, and Section 3.6 respectively.

3.1 Overview

This section describes the methodological framework to address the research questions introduced in Section 1.3. The recommender systems can be evaluated either online or offline [22]. In online evaluation, real users test the system and empirical comparisons of the user's satisfaction are done using A/B tests. Online evaluation requires a working system with a large number of users. Online evaluations are not always feasible due to the factors including the number of users and high cost, as mentioned in [21]. On the other hand, offline evaluations make use of a database of users' history of preferences to assess the performance of the system. As a result, offline evaluations are repeatable without any cost overhead. Hence we limit the use of offline evaluations in this thesis using datasets detailed in Section 3.2.

For a given offline recommendation task, the two important components according to [10] are the methodology and the metrics. The methodology is the specific way in which we instantiate the particular recommender system experiment. The generic methodology applicable to all experiments is detailed in this section. The methodology for each of research questions introduced in Section 1.3 is detailed in Chapter 4, Chapter 5, Chapter 6, and Chapter 7 respectively.

In general, a recommendation model is built using the training data obtained from the data splitting step in Section 3.1.2. Further pre-preprocessing of the training set Tr maybe required based on the recommendation framework used. The training of the recommender system model is specific to the dataset and the RS algorithm. The RS framework used in this thesis is detailed in Section 3.5. The hyperparameters are optimized as detailed in Section 3.5.1. The steps to train the recommender system are detailed in the next chapters. Once the model is built, the test set Te is used as the ground truth and the recommender is evaluated against it. The metrics measure the desired properties of the recommendations. The metrics used for evaluation is detailed in Section 3.4. Finally, once the evaluation is complete, the results are analyzed. The overview of steps involved in the offline evaluation of a recommender system is shown in Figure 3.1.

The terms and definitions that will be used thereafter in this thesis are formally defined below.

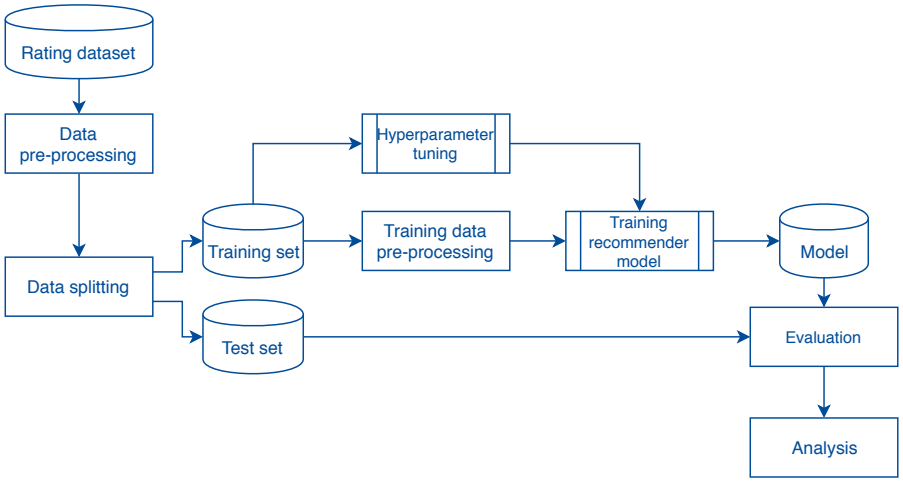


Figure 3.1: Overview of the steps in the offline evaluation of RS.

Terms and definitions

- **Ratings matrix:** Set of all user preferences denoted by R. The ratings matrix with timestamps is made up of the following four tuples $[u, i, r_{u,i}, t_{u,i}]$ where u is the user, i is the item, $r_{u,i}$ is the rating for the user-item pair, and $t_{u,i}$ is the timestamp for the user-item pair.
- **User profile size:** Count of all the items rated by the user in a given set of ratings.
- **Ordering condition:** Conditions which determine the sequence for a given set of ratings. The condition could be dependent on time, user, item, and rating.
- **Training set:** Training set, Tr is the set of all ratings used to train a RS.
- **Test set:** Test set, Te is the set of all ratings used to evaluate the performance of the RS.
- **Split:** A split Σ partitions the ratings matrix R into a training set Tr and a test set Te such that $Tr \cap Te = 0$.
- **Hold-out splitting:** Exactly one training set Tr and one test set Te without overlap $(Tr \cap Te = 0)$ is used for the evaluation of the RS.
- **Cross-validation:** One or more data splits from the given set of ratings is used for evaluation to minimize the variability of evaluation results.
- **Cold start user:** Users who have ratings in the test set Te without any ratings in the training set Tr, i.e. $u \in Te$ and $u \notin Tr$.
- **Dormant user:** Users who are part of the training set Tr without any ratings in the test set Te, i.e. $u \in Tr$ and $u \notin Te$.
- **Temporal window:** A subset of the temporally ordered ratings matrix R constitute the temporal window W. The ratings in the temporal window W are time-bound between q_{start_time} and q_{end_time}.

3.1.1 Data pre-processing

Data pre-processing is the first step in the evaluation pipeline. The experiments in this thesis rely on the temporal ordering of the ratings data. Hence the rating ordering condition based on the increasing order of time is applied to the dataset which determines the sequence of the recorded ratings. Such an ordered dataset is usually referred as temporally ordered dataset. This pre-processing step is used in all of the experiments in this thesis. In this pre-processed dataset, the temporally old ratings are at the end of the rating matrix, while the recent ratings are at the other end.

3.1.2 Data splitting

Data splitting is the partitioning of the history of user preferences into a training set Tr and a test set Te. In offline evaluation, we train the recommender system using the training set Tr and evaluate the performance against the test set Te. Since the test set Te should not be available during training, there should not be any user-item pair overlap between the sets. Data splitting process should ensure that $\forall\ r_{u,i} \in R, (r_{u,i} \in Tr) \cap (r_{u,i} \in Te) = 0$.

The splitting of the dataset is usually based on certain criteria. There are a number of well defined methods for data splitting [21, 22, 10]. Time based threshold conditions such as q_{start_time} and q_{end_time} can be used to extract specific data for the training and test sets. In the case of temporally ordered data, the partition boundary or the split point could be a date or a rating with a timestamp. The selected rating, r_s forms the partition boundary splitting the training and test sets. All the ratings prior to the selected rating r_s form the training set Tr. The remaining ratings form the test set Te. The partition boundary could be selected to vary the ratio of ratings in training and test sets.

Not all users and items are represented in both the training and test sets. Temporal data splitting results in the formation of cold-start users and items. These are the new users and items that becomes part of the dataset after the temporal split point will have no historical ratings in the training set Tr. Similarly, users who have been inactive for a time will have historical preferences in the training set. These users who are not represented in the test set and will not be evaluated are called as dormant users. The dormant users have a small impact on the overall performance of the RS as they are used to model future predictions of the test set Te. On the other hand, the cold-start users adversely affect the performance of the recommender system as they are not modeled at all.

3.2 Datasets

The datasets used for evaluating recommender system can be broadly classified into two based on the type of rating as *explicit*, and *implicit* datasets. We use the classic explicit dataset, MovieLens [1] as the primary dataset for our experiments and is described in the following section.

MovieLens dataset

MovieLens is an online movie recommender system run by the University of Minnesota's GroupLens[2] research team. Their publicly available MovieLens datasets [23] are widely used in academia and they are suitable for comparing results across the literature. In this thesis, the explicit ratings along with timestamp are used in experiments and MovieLens dataset is a good candidate dataset for experimentation. Auxillary information such as movie genre, user tags, etc., are not used as part of the experiments.

The first MovieLens dataset [34], MovieLens 1 million ratings (ML-1M), was released in 1998. Two more milestone versions were released: MovieLens 10 million ratings

[1] https://www.movielens.org
[2] https://www.grouplens.org

Table 3.1: MovieLens dataset statistics.

Dataset	Date Range	Users	Movies	Ratings	Density
ML-10M	1/1995-1/2009	69,878	10,681	10,000,054	1.34%
ML-20M	1/1995-3/2015	138,493	27,278	20,000,263	0.54%
ML-6M	1/1999-9/2006	36,804	9,132	6,600,000	1.94%

(ML-10M) in the year 2005, and MovieLens 20 million ratings (ML-20M) in the year 2016 respectively. The ML-10M and ML-20M datasets capture the explicit preferences of the users' rating of the movie on a 0.5-5 scale with 9 possible discrete ratings for each movie along with the timestamp. The timestamp denotes the time of the user rating the movie and not that of the actual consumption of the movie.

The MovieLens dataset can also used as a dataset with implicit feedback. The user's explicit ratings can be converted into implicit feedback in one of the following ways. In [25], Hu et al. consider the movies rated by the user as positive feedback and the movies not rated by the user as negative feedback. Most times, a threshold is set for selection of positive feedback from the ratings. The ratings above the threshold are taken as positive feedback and the rest are considered as negative feedback.

The ML-20M dataset is an extension of the ML-10M dataset. While the ML-10M dataset spans over a period of 14 years, the ML-20M dataset covers a span of 20 years and includes the ratings of ML-10M as well. All users in the dataset have a minimum of 20 ratings. The statistics for the MovieLens dataset are shown in Table 3.1. While the number of ratings and the number of users are doubled in the ML-20M dataset, the number of items is increased by a factor of 2.5. As a consequence, the density of the ratings in the ML-20M dataset reduces resulting in a very sparse dataset. In this thesis, we limit the scope of the experiments to ML-10M dataset. The ML-6M dataset is a subset of ML-10M dataset as detailed in Section 5.2.1. The characteristics of the MovieLens dataset is discussed further in the following sections.

3.3 Analysis of MovieLens 10 million dataset

The characteristics of the recommender system change over time as new users and items are added to the system. Apart from the users and items, the newer rating input into the system by the users has an impact on the overall performance of the recommender system. This section focuses on the changes that occur in the dataset over time and its influence on the conclusions drawn. The temporal characteristics of a temporally pre-processed ML-10M are discussed in the following section. Firstly we focus our analysis on the growth of the number of users, items and ratings over time. Then we analyze the user's profile sizes by binning them.

Growth over time

For the ML-10M dataset, the cumulative growth of the users, items, and ratings dataset are visualized in Figure 3.2, Figure 3.3, and Figure 3.4 respectively. The new users, items, and ratings added to the system on a weekly basis is shown in Figure 3.5, Figure 3.6, and Figure 3.5 respectively. We group the new ratings weekly to identify trends and sharp changes in activity levels recorded in the dataset.

Using both the cumulative and new addition graphs for the items as shown in Figures 3.3, 3.6, we can concur that the number of items in the dataset increases more or less constantly. New items are added at a constant level. Though there are a few spikes in Figure 3.6, the total number of items added per week is spread when compared to users and ratings.

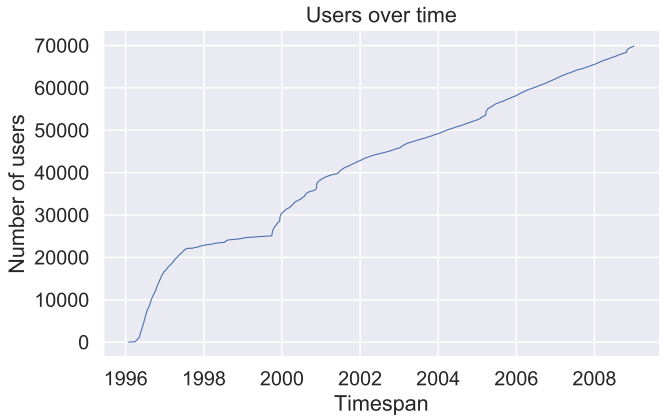


Figure 3.2: Number of users over time for the ML-10M dataset.

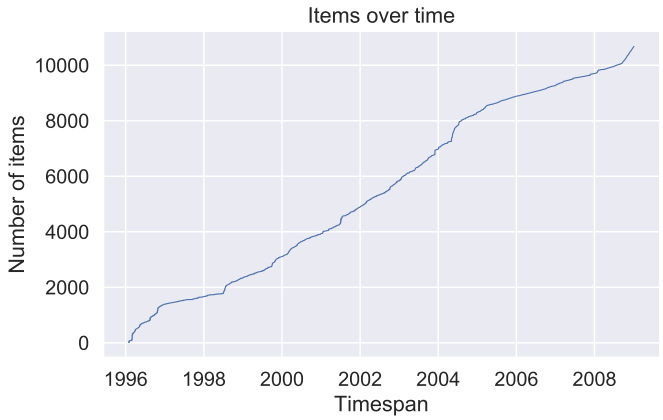


Figure 3.3: Number of items over time for the ML-10M dataset.

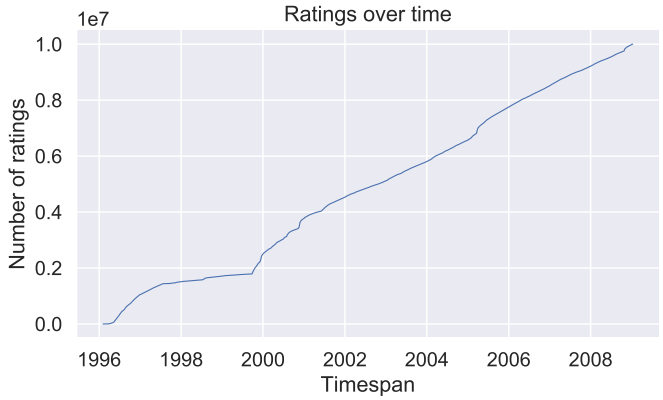


Figure 3.4: Number of ratings over time for the ML-10M dataset.

Using both the cumulative and new addition graphs for the users as shown in Figures 3.2, 3.5, we can spot growth sprouts at certain points including the growth around the years 2000 and 2005. The growth of the ratings dataset follow a similar pattern and it clearly shows that the growth in the number of users and the number of ratings fed in to the system are related.

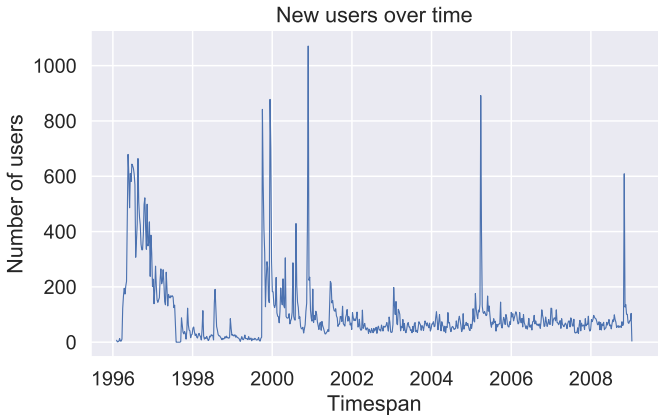


Figure 3.5: New users over time for the ML-10M dataset.

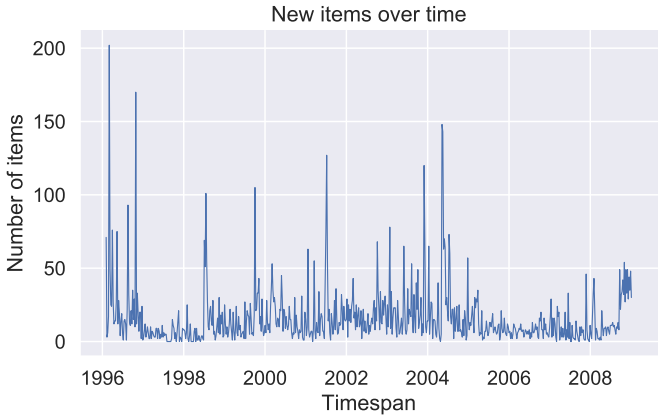


Figure 3.6: New items over time for the ML-10M dataset.

At these growth sprouts, the number of users in the system rise sharply and could affect the performance of the recommender system. For instance, the sudden increase in number of users increases the cold-start users and adversely affects the performance. Though there are a few spikes in Figure 3.6, the total number of items added per week is spread when compared to users and ratings. The number of new ratings recorded per day is shown in Figure 3.7. The date of the last rating by a user i.e. the date on which the users stopped using the system is illustrated in Figure 3.8. The users who have stopped using the system are called inactive users.

Size of user profiles

First, the size of a user's profile is computed by counting the number of items rated by the user. The ML-10M dataset contains a minimum of 20 ratings for all the users. Figure 3.9 compares the cumulative percentage of users and the size of their profiles. We observe that the number of users with shorter profiles near to the minimum profile size of 20 is exponentially high. 25% of users have rated less than 35 ratings, and 50% of users have rated less than 70 ratings.

In Figure 3.10, we plot the sizes of all the user profiles grouped in bins in the range of [0,500], [500,1500], and 1500+ ratings. By dividing Figure 3.10 into three sections in the range of [0,500], [500,1500], and 1500+ ratings, we get a better clarity of the distribution of users with long profiles. 13,565 profiles have more than 200 ratings. There are only 97

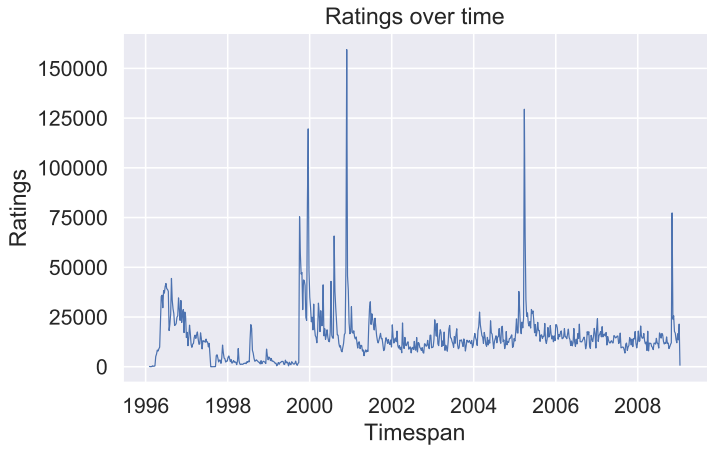


Figure 3.7: New ratings over time for the ML-10M dataset.

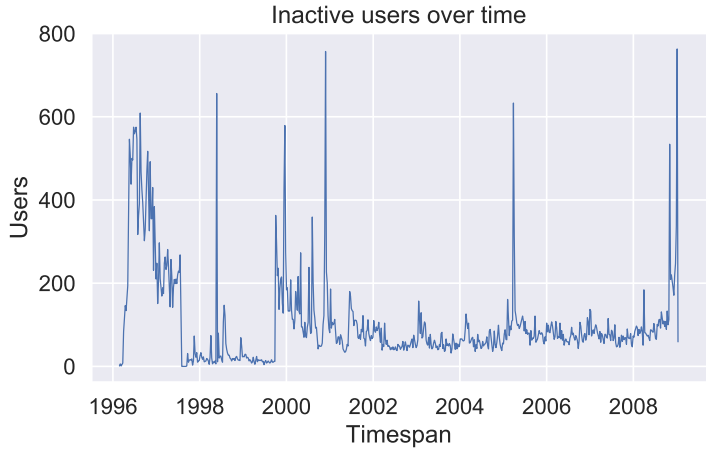


Figure 3.8: Count of users who have stopped rating new items in the ML-10M dataset.

profiles with more than 2000 ratings, even though there are 837 profiles with more than 1000 ratings. There exist 7 large profiles with more than 4000 ratings.

3.4 Evaluation metrics

The quality of a recommender system [22] can be determined by a number of properties including Prediction Accuracy, User Preference, Coverage, Diversity, Serendipity, Novelty, etc., based on the experimentation conditions. In order to evaluate the offline experiments in our thesis, we test the prediction accuracy of the recommender system. The literature on the recommender system classifies prediction accuracy into two: rating prediction and ranking [42] based on the recommendation task. As discussed in Section 3.2, we limit the scope of the experiments to the rating prediction. The different metrics used for measuring the accuracy of the rating prediction task in RS is briefed in this section.

3.4.1 Measuring the accuracy of rating predictions

Rating prediction is the recommendation task aiming to predict the rating $\hat{r}_{ui}$ for items that are yet to be rated. The system predicts for a test set Te for which the true ratings, r_{ui} are not

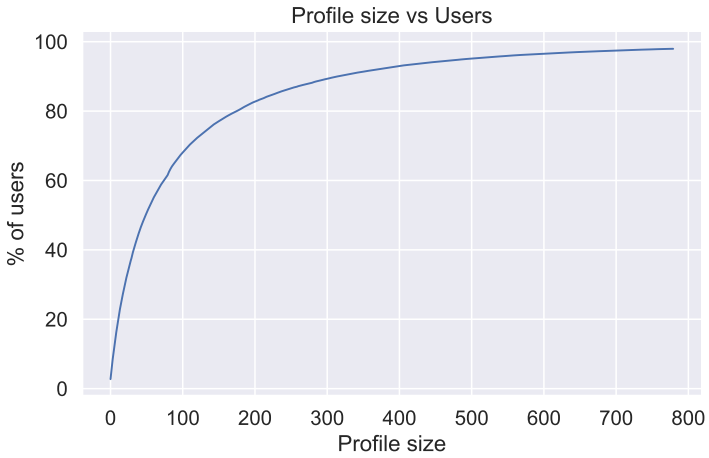


Figure 3.9: User profile sizes vs cumulative % of users.

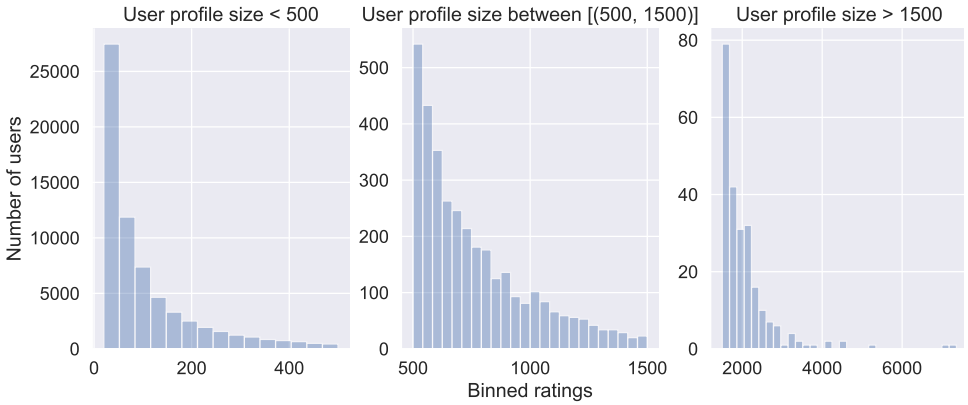


Figure 3.10: User counts grouped by profile size.

known. For a single prediction task, the error is the difference between the predicted rating $\hat{r}_{ui}$ and the true rating r_{ui}. For numerical rating predictions, Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) are two of the most popular metrics. MAE is the mean of all the absolute errors in the test set *Te*. RMSE is the square root of the mean of the square of all the errors in the test set *Te*. MAE and RMSE and are defined as follows:

$$MAE = \frac{1}{|\,Te\,|}\Sigma_{(u,i)\in Te} \mid \hat{r}_{ui} - r_{ui} \mid \tag{3.1}$$

$$RMSE = \sqrt{\frac{1}{|\,Te\,|}\Sigma_{(u,i)\in Te}(\hat{r}_{ui} - r_{ui})^2} \tag{3.2}$$

Though both the above metrics MAE and RMSE measure the difference between predicted and true ratings, they differ in the degree of error penalties. RMSE disproportionally penalizes large errors when compared to MAE. In this thesis, RMSE is used as the primary metric for measuring the prediction performance.

3.4.2 Measuring the accuracy of ranking of items

Ranking metrics are useful when the recommendation task is to pick a set of items, *Te*, for a user *U* among all available items in the dataset. Ranking metrics are widely used along with implicit datasets where the interaction between the user and the item is recorded without a numerical rating. The use of the ranking metrics is outside the scope of this thesis.

3.5 Recommender framework

As discussed in Section 2.3, we use BMF algorithm for recommendations. Said et al. [39] highlights the differences in the implementation of similar algorithms with differences in reported recommendation quality. They present the need for greater detail recommender algorithms' tuning and results for a fair evaluation. To produce fair comparison and reproducible results, we follow the recommendations of Said et al. [39] by presenting as much of the implementation details as possible. In this section, we detail the hyperparameter tuning of BMF and discuss the baseline results of BMF.

3.5.1 Hyperparameter optimization for BMF

In general, the hyperparameters are tuned to maximize the performance of the algorithm using the training set. The training and test sets need to be mutually exclusive as the system must not use any of the data in the test set for tuning. Optimization of the hyperparameters is a vital step in the machine learning pipeline to obtain good results. The models trained with the optimized hyperparameters are evaluated on the test set to measure the performance. One widely used method is to sample a portion of the training set as the validation set using which the parameters are optimized. The other option is to use k-fold cross-validation on the training set.

The tunable hyperparameters, which have significant impact on the performance of the chosen CF algorithm– BMF, are *the learning rate (lr)*, and *the number of factors (nf)*. For the experiments in this thesis, the hyperparameters are tuned using cross-validation and grid search against the RMSE values. During the tuning process, several hyperparameter values that were unlikely to produce optimized results were skipped. The hyperparameters were tuned for 100 iterations using the following set of values with experiment specific exclusions:

- Number of factors *nf*: [8, 10, 15, 20, 30, 50]
- Learning rate *lr*: [0.0005, 0.00075, 0.001, 0.00125, 0.00175, 0.0025, 0.005, 0.01, 0.015, 0.02]

The tuning of the hyperparameters is depended on the dataset. As the primary focus of this thesis is on the training data analysis, the training dataset varies from one experiment to the other. As a consequence, the validity of the chosen hyperparameter for a specific set of experiments may not be good for the other. Hence the validity of the hyperparameter is asserted for experiments under each of the research questions. The experiment specific tuning of the hyperparameters are described in Section 4.2.1, and Section 5.2.2.

3.5.2 Baseline results for BMF

In this thesis, we do a comparative study of the different techniques described in Section 1. Baseline results help to systematically compare the results of the different experiments in a principled manner. The baseline experiment uses the entirety of the ML-10M dataset. The default set of hyperparameters, that give the best results for BMF are used. The learning rate is set at 0.001 with 20 latent factors. Five-fold cross-validation is used to determine the performance metric for rating prediction, RMSE. The baseline results in terms of RMSE for the BMF algorithm using the above parameters is 0.79.

3.6 Tools

The experiments in this thesis make use of some open source algorithms and third-party tools. The details of the different tools and frameworks used are described in this section.

MyMediaLite

MyMediaLite [3] is an open-source recommender system framework developed at The University of Hildesheim [19]. MyMediaLite is widely used by researches both in academia and industry. Consequently, a large corpus of published results can be used for comparison across the literature. This framework implements several recommender system algorithms in C#. In addition to the recommendation algorithms, it also provides the evaluation routines for both prediction and ranking tasks. This thesis uses MyMediaLite's implementation of *biased matrix factorization* for experiments.

Environment

This work was primarily carried out on the Dutch national e-infrastructure with the support of SURF Cooperative. The computational power for the experiments was provisioned on the SurfSara HPC cluster using 64-bit Linux virtual machines. Some of the experiments were computed on the high performance computing cluster of Intelligent Systems Department, University of Delft.

Source code

The source code for this thesis will be made available via a git repository [4] upon the publication of this report. Python is the main programming language used in this thesis. Anaconda[5] distribution of python 3.7 is used for data analysis, preprocessing, experimental setups, and data visualization. Some frequently used modules of note include numpy, scipy, pandas, and jupyter. Matplotlib based visualization library, seaborn[6], was used in visualizations.

[3] http://www.mymedialite.net
[4] https://github.com/kman0/msc-thesis-src
[5] https://anaconda.com
[6] https://seaborn.pydata.org/

Chapter 4

Temporal window truncation

This chapter addresses the first data minimization technique formulated in the research question, RQ1, in Section 1.3. We study the effects of reducing the training data of the RS using temporal windows. First, we give an overview of the research question. Then we detail the methodology used to answer the research question in Section 4.1. The experiment is described in Section 4.2 followed by the results and discussion in Section 4.3 and 4.4 respectively.

The first data minimization technique addresses the following research question:
RQ1: *How does the temporal window truncation of training data affect the performance of the recommender system?*

4.1 Methodology

The first data minimization technique, temporal window truncation, aims to understand the impact on the performance of the BMF algorithm by varying the size of the training data in terms of temporal windows. In a temporally ordered ratings matrix, the set of ratings between two ratings or two points in time constitute a temporal window. Training data analysis assesses the impact of the performance of the BMF algorithm by varying the history length of the windowed training data.

The temporally ordered ratings matrix, $R_{ordered}$, is segmented into N equal windows. Unlike [31] where eleven splits are used, we set it $N = 10$. The temporally last window that contains the recent ratings is the test set and the window preceding it as the training set. The remaining data are discarded. The performance of the BMF algorithm is evaluated for this window. In each of the successive iterations, the number of windows in the training data is increased by adding one more historical window (window adjacent to the training window) of ratings data. It is important to note that all the training sets use the same test set for comparability. This is illustrated in Figure 4.1.

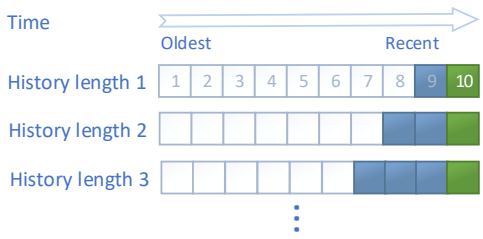


Figure 4.1: Training data analysis by increasing the history length of the windowed training data.

4.1.1 Sliding window cross-validation

Cross-validation (CV) is a common method used for the evaluation of a recommendation algorithm to flag problems, including over-fitting. Cross-validation repeats the evaluation process several times with different data splits. Some common CV techniques used in the literature include repeated sampling hold-out validation, user re-sampling hold-out validation, k-fold cross-validation, and leave-one-out cross-validation [21]. Traditionally, CV does not differentiate training vs. test as it makes use of the entire dataset, ignoring the time dependencies in the training and test sets[11, 45] and have an unfair advantage in predicting a target user's recommendation [11]. For example, in the case of five-fold cross-validation in collaborative filtering where random splits are used, the recommender uses the future preferences of other users to predict recommendations for the target user. As the k-fold cross-validation violates the ordering of data, the evaluation techniques used in time series are considered. To overcome the above drawbacks of cross-validation in recommender systems, Gordea et al. [20] introduced varying window experimentation.

The varying window experimentation is based on forward chaining, which used in time series evaluation and is illustrated in Figure 4.2. The windows are created by partitioning the dataset into distinct, temporally ordered windows. In their experiments, the size of the test window is constant, and the position of the test window rolls forward. In successive iterations, the segmented train data and test data rolls forward to create new data splits. The train data now includes one more window of data, which was previously the test data in the previous iteration. Nevertheless, the test set is always the most recent window and changes with each roll. Comparing the performance of RS algorithm across training sets of varying sizes is not fair as the characteristics of the training set play an essential role in the performance of the chosen RS algorithm as seen in Section 3.5.1. This drawback is overcome by the use of sliding window cross-validation introduced by Larson et al. [31].

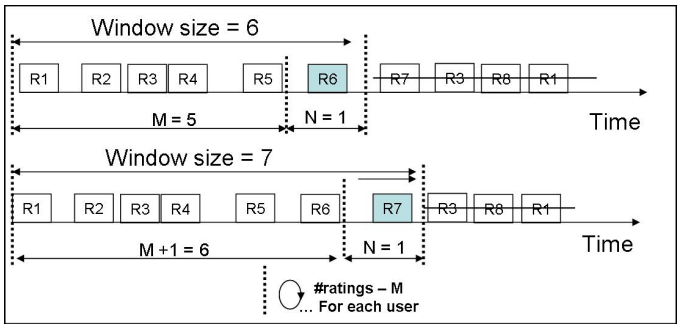


Figure 4.2: Varying window experimentation by Gordea et al.[20].

Sliding window cross-validation method [31] computes the average performance across multiple folds of the dataset with overlapping subsections of the data for training and evaluation. In general, a fold is a virtual partition of the original dataset that is constructed based on certain criteria. In the case of sliding window CV, the first fold is constructed by using the latest window as the test set and the preceding windows as training set. The second fold is constructed by using the latest window of the previous fold's training set as the test set and its preceding windows as the training set. The data after the test window is discarded from the fold. Similarly, k folds can be constructed by following the above function. A 3-fold sliding window CV is illustrated in Figure 4.3.

The sliding window CV's main difference with the variable window experimentation [20] is the use of the training set of constant size. This reduces the variability in the evaluation of the RS that depend on the size of the training set. However, it does not account for underlying changes in characteristics of subsets of data other than the size of the training sets. Unlike the variable window experimentation where the training data window is slid

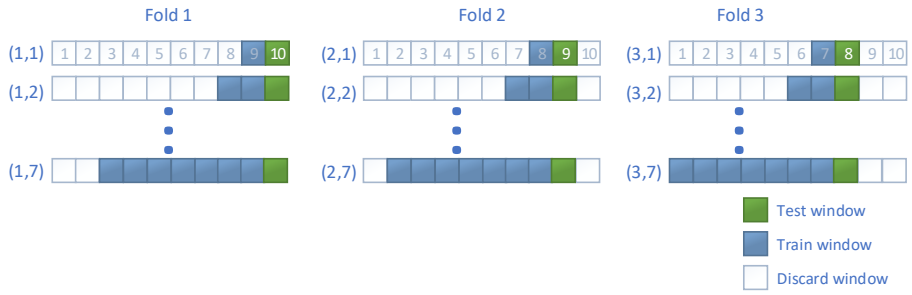


Figure 4.3: 3-fold sliding window cross-validation

Figure 4.4: Variable window experimentation vs 3-fold sliding window cross-validation

forward, the training data window is slid backward in the case of sliding window CV to keep the size of the training set constant. In both cases, the test segment is always the temporally most recent segment. The difference between variable window experimentation and sliding window cross-validation is illustrated in Figure 4.4.

4.1.2 Temporal window splitting

Algorithm 1 defines the function, *generate_temporal_data*, which is used to generate the training and test sets for the experiments. For a given fold $fold$, and length of history windows len, maximum number of window segments, N, and the ratings matrix R, the function *generate_temporal_data* generates the training set Tr and the test set Te. The training set is identified by the combination of the fold number and its history length– $(fold, len)$. The maximum number of window segments, N, is used to divide R into N segments and is defaulted to 10. In the case of the MovieLens 10 million ratings (ML-10M) dataset, each window segment consists of exactly 1 million ratings.

4.2 Experiment

In this section, we define the experiments for the temporal window truncation of ratings data in order to answer the proposed research question in this chapter. First, we provide an overview of the experiment. Then, the algorithm used to generate data splits for the training and the test is in Section 4.1.2. The selection of hyperparameters is briefed in Section 4.2.1 and is followed by the experimental setup in Section 4.2.2.

In 3-fold sliding window cross-validation, we generate different data for 3 folds. The temporally latest window, window-10, is used as the test set for the first fold, fold-1. Similarly, for fold-2 and fold-3, the test windows are window-9, and window-8 respectively. This is illustrated in Figure 4.5. For the last fold, the maximum number of windows that

Algorithm 1: Generate the training and the test set for temporal window truncation.

```
1  function generate_temporal_data (R, fold, len, N = 10);
   Input  : Ratings matrix R
            Fold fold
            Length of history windows len
            Maximum number of windows in R N
   Output : Trainingset(Tr), Testset(Te)
2  Tr ← ∅
3  Te ← ∅
4  W_i ← ∅
5  R_ordered ← Sort R by time
6  window_len ← length(R)/N
7  for i ← 0 to N do
8      start ← i * window_len
9      W_i ← [R_ordered]_{i=start}^{window_len} ;          // extract windows
10 Te ← W[N − fold]
11 start ← N − len − fold
12 end ← N − fold
13 Tr ← Σ [W_i]_{i=start}^{end} ;                  // concatenate the windows
14 return Tr, Te
```

can constitute the training set is 7. Hence we limit the maximum size of the training set for fold-1 and fold-2 to 7 windows. Ratings in the windows before and after the selected training and test sets are discarded. For a given fold, the test set remains constant across all training data sets of varying history lengths. We generate seven sets of training data for each of the three folds resulting in 21 sets of training data and 3 sets of test data. The function to generate the training and the test sets are detailed in the following section.

Figure 4.5: Sliding window cross-validation across 3 folds of history length 7.

For each of the three folds, seven different training sets with an increasing number of windowed history lengths with a maximum windowed history length of 7 are constructed. The shortest training set of history length one contains exactly one train and test segment. The shortest training sets are (1,1), (2,1), and (3,1). The longest training sets are (1,7), (2,7), and (3,7), which contains 7 segments in train set and 1 segment in test set. In all, 21 training sets and three distinct test sets are generated for experimentation. This is shown in Figure 4.3.

4.2.1 Hyperparameter optimization

The hyperparameters were tuned for BMF algorithm in Section 3.5.1 for the ML-10M dataset. In this experiment, the size of the data splits vary significantly. As a consequence, the characteristics of the training set and test set for the experiments vary drastically depending on the size. The validity of the previously tuned hyperparameters is questionable. Hence we tune for the hyperparameters again using the training set as detailed in Section 3.5.1. First, we tune the learning rate lr over a range of 0.0005 and 0.02. The results are shown in Figure 4.6. The optimized value for learning rate, $lr = 0.001$ is used for experiments in this section.

Similarly, the number of factors used for training was varied from 8 to 50 as shown in Figure 4.7. The set of values used for the tuning process is detailed in Section 3.5.1. Based on the results of hyperparameter tuning, we observe that for the number of factors have little impact on the performance of the BMF across the folds. However, the time for training increases as the number of factors increases. Hence we limit our experiments to use 20 latent factors, which produces good results for most of the folds. The optimized value for the number of factors, $nf = 20$ is used for experiments in this section.

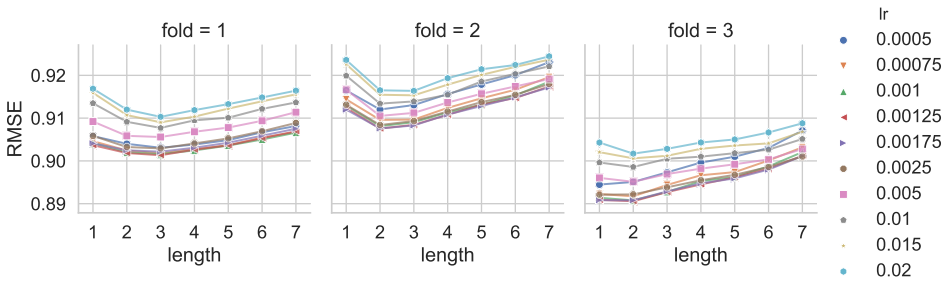


Figure 4.6: Hyperparameter optimization on the learning rate for *nf=20*.

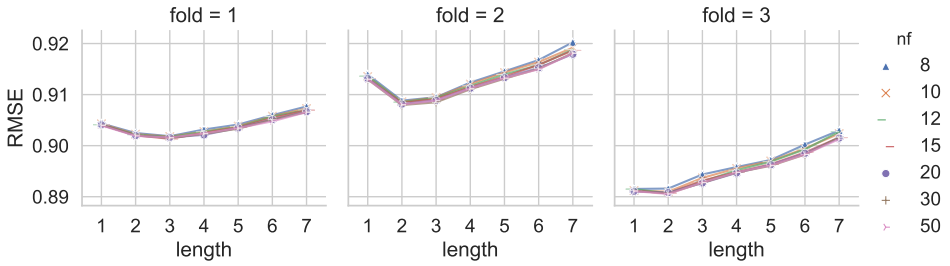


Figure 4.7: Hyperparameter optimization on the number of factors for *lr=0.001*.

4.2.2 Experimental setup

The training data analysis experiments are performed by evaluating the performance of the BMF algorithm with training data of varying history lengths against a constant test set. The BMF is trained using the optimized hyperparameters: $nf = 20$, and $lr = 0.001$. The tuning of hyperparameters are detailed in 4.2.1. The evaluation metric, RMSE, is measured for 21 different training sets. The test set is the same for all the training sets in the same fold. The test set contains 8288, 7922, and 7533 users and 10116, 9225, and 8394 items in each of the folds. Since we are using sliding window CV, the average performance of the training sets of the same history length is computed across the folds as shown in Figure 4.5.

Table 4.1: Windows history lengthwise statistics of the generated data per fold.

Length	Fold	Items ∈ *Tr*	Users ∈ *Tr*	Items ∈ *R*	Users ∈ *R*	Density
1	1	9,225	7,922	10,514	13,639	1.39%
	2	8,394	7,533	9,382	12,788	1.67%
	3	8,128	7,161	8,742	12,100	1.89%
2	1	9,382	12,788	10,570	18,326	1.55%
	2	8,742	12,100	9,464	17,183	1.84%
	3	8,465	12,163	8,924	16,891	1.99%
3	1	9,464	17,183	10,603	22,660	1.66%
	2	8,924	16,891	9,539	21,875	1.92%
	3	8,520	17,798	8,964	22,465	1.99%
4	1	9,539	21,875	10,650	27,326	1.72%
	2	8,964	22,465	9,559	27,413	1.91%
	3	8,533	23,378	8,975	28,035	1.99%
5	1	9,559	27,413	10,667	32,854	1.71%
	2	8,975	28,035	9,569	32,971	1.90%
	3	8,544	29,083	8,983	33,734	1.98%
6	1	9,569	32,971	10,669	38,408	1.71%
	2	8,983	33,734	9,574	38,660	1.89%
	3	8,553	40,063	8,989	44,711	1.74%
7	1	9,574	38,660	10,673	44,093	1.70%
	2	8,989	44,711	9,579	49,636	1.68%
	3	8,553	54,873	8,989	59,521	1.50%

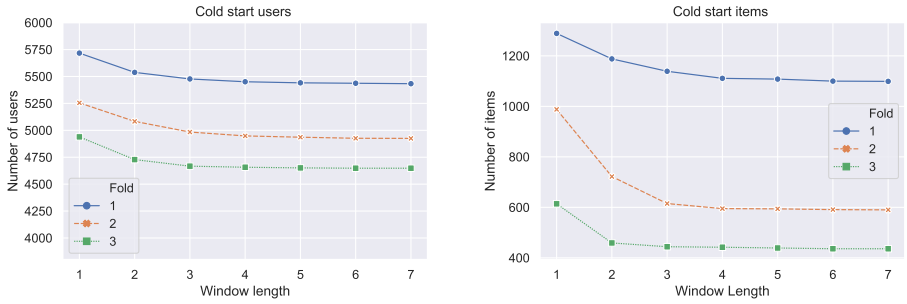
4.3 Results

The characteristics of the generated training and test data sets are shown in Table 4.1. The table lists the following statistics for all of the generated rating matrices: the total number of users in *R*, the number of users in *Tr*, the number of items in *R*, the number of items in *Tr*, and the density.

From the Table 4.1, we observe that the number of items in the training set T_r for a given history length is the lowest in fold-3, and increases gradually in fold-2, and fold-1. The number of items varies minimally across the different history lengths. This shows that the items have good coverage in training sets. The total number of items in the ratings matrix *R* follows a similar pattern. The changes in the number of cold-start items are captured in Figure 4.8b. The number of cold-start items per fold decreases gradually up to the history length of 4 and stabilize after that. The number of items that were not evaluated increase sharply for the first three windowed history lengths and stabilize after that as seen in Figure 4.9b.

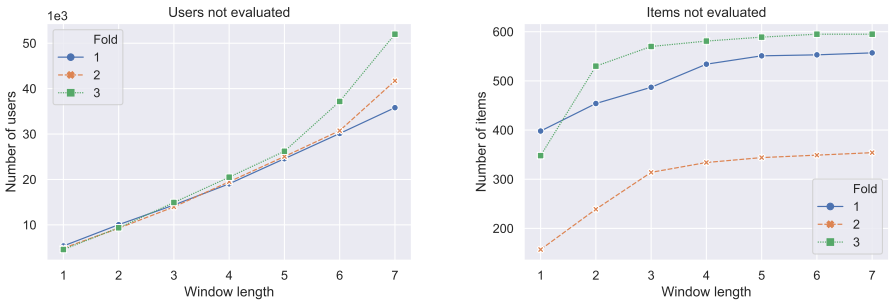
The number of users in the ratings matrix *R* and the training set T_r increase sharply with increasing history lengths. This directly correlates with the increase in the number of users as discussed in Section 3.3. The variance in the number of users between folds for a given history length is minimal when compared to the overall increase in the number of users. The changes in the number of cold-start users are captured in Figure 4.8a. The number of cold-start users decrease for the first three history lengths and stabilize after that. The number of users that were not evaluated increase sharply for all the folds with increasing history lengths as shown in Figure 4.9a.

The changes in the density of the ratings matrix *R* of all the generated data is captured in Figure 4.10. The density of the ratings matrix increases for the first three windowed history lengths for all the folds. For all the folds, the density is at maximum i.e. the sparsity of the ratings matrix is at its lowest for the windowed history lengths of 4 and 5. Then the density decreases for the folds fold-2 and fold-3 at history lengths of five and six. This



(a) Cold-start users. (b) Cold-start items.

Figure 4.8: Number of cold-start users and items for different folds of data.



(a) Users not evaluated. (b) Items not evaluated.

Figure 4.9: Number of users and items that were part of training data but were not evaluated.

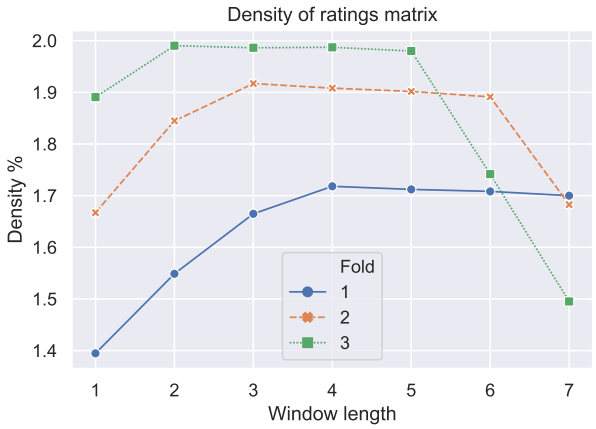


Figure 4.10: Changes in density of the ratings matrix R per fold and history length.

correlates with the steep increase in the number of users and items at the beginning of the full ML-10M dataset.

The overall performance and the performance of cold-start users of the BMF algorithm for the 3-fold sliding window CV measured in terms of RMSE is shown in Figure 4.11. We observe that the overall performance slightly decreases for the first two folds and it increases after that. The cold-start performance The per fold performance of the 3-fold sliding window cross-validation is shown in Figure 4.12. The three folds show different performance trends. Since the test data of the different folds are not the same, the performance comparison across

folds is not fair. The performance of the fold-2 follows the trend of the cross-validated overall performance in Figure 4.11. The variance in RMSE across folds is minimal for the fold-2. The performance of the fold-3 increases with an increase in the length of the windows.

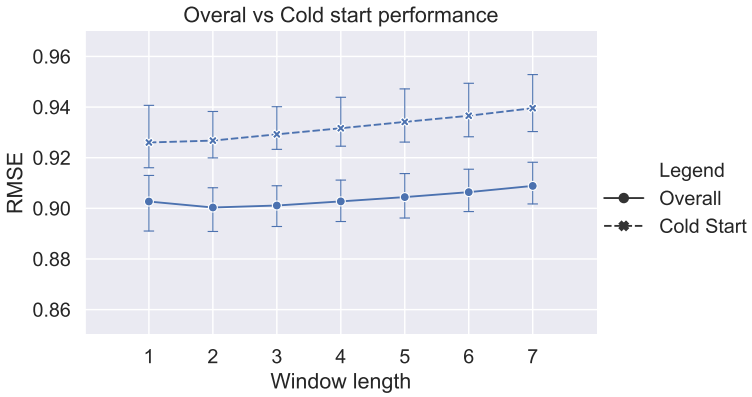


Figure 4.11: Performance of 3-fold sliding window cross-validation.

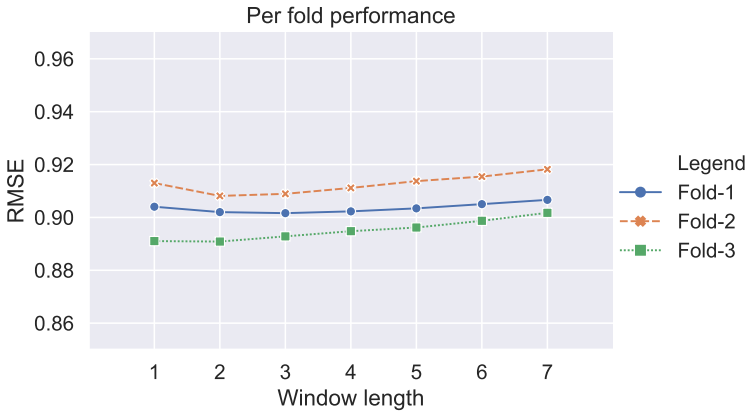


Figure 4.12: Comparison of the different folds of the sliding window cross-validation.

4.4 Discussion

The experiments show the effects of varying the size of the training data in terms of temporal windows. When we increase the length of the temporal window, the underlying characteristics of the training data set changes. The findings show that the overall performance in terms of RMSE increases for the window length of two. Then the RMSE increases as the size of the training set increases. More data points do not contribute to the increase in performance. We observe that the overall performance of the BMF decreases after window size of three.

This experiment is based on the earlier work done by Larson et al. in [31] with differences in the number of folds used for sliding window cross-validation and the number of ratings per window. The overall RMSE metrics per window length are higher in our case. This observation could be due to the increase in the number of ratings for the same window length resulting in a larger test set. We confirm that the trend in the performance of the BMF algorithm obtained here is similar to the work done in [31].

On closer analysis using the per fold performance from Figure 4.12, we can see that the folds two and three show considerable increase in RMSE when compared to fold-1. This could be attributed to the timeliness of the data as these folds use temporally older data containing more users. The sparsity of the window of length 6 and fold-2 and the window of length 7 of fold-2 and fold-3 increases and this directly correlates with the decrease in performance. Also, pruning of the inactive users as seen in Figure 3.8 could have an impact. Fold-3 has the most users and items that are not evaluated as illustrated in Figure 4.9a and Figure 4.9b respectively when compared to the other folds. On the other hand, the inactive users per window length are the least in fold-1 and as a result, the RMSE decreases.

The experiments use strict temporal splitting of training and test sets of fixed size. As a result, a large number of new users irrespective of their profile size become cold-start users. Moreover, new items that are introduced into the system after the split point are cold-start items. This is likely the reason why the number of cold-start users and items are very high for the fold-1 containing the latest ratings data. Likewise, users with short profiles are more likely to be part of the test set affecting the performance of the RS. Detailed analysis of the impact of such users is required for a better understanding of the impact of the cold-start users on data reduction.

As the size of the window increases, the number of users who are not evaluated increases. This increase in users decreases the density. As the test set is common for all window sizes, the number of cold-start users and items decreases. The observed differences in performance could be due to the inadvertent pruning of inactive users as a direct consequence of window truncation. This prompts further research.

This experiment only considers the effects of BMF. By evaluating the above setup for CF algorithms other than BMF could shed light on the algorithmic influence of BMF. Though the similar work [31] uses factorization machines in addition to BMF, using more algorithms for experiments is warranted. This is a known limitation for all the experiments in this thesis. We would like to work on evaluating the experiments using different CF algorithms in the future.

The temporal effects of the underlying data such as drift, decay, seasonality, and biases as detailed in [10] in these temporal windows are not considered. Evaluating the above setup with CF algorithms that take temporal effects would help to understand the usefulness of the proposed data minimization method. For instance, modifying the decay function in a temporal CF algorithm to use a step function could mimic the effects of time truncation. This might help us to formulate and measure the effectiveness of data minimization directly rather than using the performance of the RS. This could have direct implications on data retention policies and in turn enhance the privacy of the users.

This experiment does not consider the distribution of ratings and items in the temporal windows i.e. the length of the user-profiles and the popularity of the items. To study the impact of user distribution further, we propose the second minimization technique, user profile truncation, in Chapter 5. More insight into the underlying temporal effects could be understood by using the differential data analysis technique [47]. Incorporating the differential data analysis with a focus on the impact of item popularity is another direction in which future work can be done.

In this chapter, we analyzed the effects of temporal window based data reduction. We underline the need for further research to study the impact of reducing the training data. We follow up the first data minimization technique, time window truncation, with with the second technique–user profile truncation in Chapter 5.

Chapter 5

User profile truncation

This chapter proposes the second data minimization technique to address the research question RQ2. First, we introduce the research question. Then the methods used to answer the research question are detailed in Section 5.1. The design of the experiments are briefed in Section 5.2, followed by the results in Section 5.3. Finally, the findings of the experiments are discussed in Section 5.4.

The second data minimization technique addresses the following research question:

RQ2: *How does the truncation of users' historical rating profiles affect the performance of a recommender system?*

5.1 Methodology

The second minimization technique, user profile truncation, is applied on a temporally ordered ratings matrix R. It is well established that historical ratings near the target recommendation data i.e. the recent ratings are of more importance for the RS [15, 16, 10].

User profile truncation of a user for the given length N retains the recent ratings N ratings for the user and truncates the rest. We perform training data analysis using different profile truncation lengths to determine the impact on the performance of the RS. Unlike the temporal window truncation where some users are removed from the ratings matrix, this technique represents all the users in the system. Users with profiles shorter than the target truncation length are not impacted with this truncation. The user profile truncation is detailed in the next section.

5.1.1 User profile truncation

Truncated training set $Tr_{truncated}$ is generated from the Tr_{full} by discarding some of the historical ratings from the users' profile based on the target profile length $truncate\_len$. The function $get\_longer\_profiles$ defined in Algorithm 2 extracts the set of users whose profile lengths are longer than the minimum length min_len.

The function to truncate the users is defined in Algorithm 3. First, the list of long user profiles, U_{long} are extracted using $truncate\_len$ using the minimum length $truncate\_len$. The profiles that are not in U_{long} are shorter profiles and are directly added to the target training set $Tr_{truncated}$. Profile truncation is then applied to all the users who are part of U_{long}. Finally, the truncated profiles are added to $Tr_{truncated}$. After truncation, $Tr_{truncated}$ contains users whose profiles have a maximum of $truncate\_len$ ratings. The truncation process does not affect the number of users in the training set $Tr_{truncated}$.

Algorithm 2: Extract long user profiles.

```
1 function get_longer_profiles (Tr_full, min_len)
    Input  : Temporally ordered, full training set Tr_full
             Minimum length of a user's profile for exclusion min_len
    Output : Longer user profiles U_selected
2 U_selected ← ∅
3 U_unique ← {user}, ∀user ∈ Tr_full
4 foreach user ∈ U_unique do
5     Profile ← Tr_full[user] ;                  // extract the user's profile
6     profile_len ← length(Profile)
7     if profile_len > min_len then
8         U_selected ← U_selected||user
9 return U_selected
```

Algorithm 3: Generate the training set with user profile truncation.

```
1 function truncate_user_profile (Tr_full, truncate_len)
    Input  : Temporally ordered, full training set Tr_full
             Maximum length of a user's profile after truncation truncate_len
    Output : Truncated Training Set(Tr_trucated)
2 Tr_trucated ← ∅
3 U_unique ← {user}, ∀user ∈ Tr_full
4 U_truncate ← get_longer_profiles(Tr_full, truncate_len)
5 foreach user ∈ U_unique do
6     Profile ← Tr_full[user] ;                  // extract the user's profile
7     if user ∈ U_truncate then
8         end ← length(Profile)
9         start ← end − truncate_len
10        Profile_trucated ← [Profile_i]_{i=start}^{end} ;  // truncate the user's profile
11        Tr_trucated ← Tr_trucated||Profile_trucated
12    else
13        Tr_trucated ← Tr_trucated||Profile
14 return Tr_trucated
```

5.2 Experiment

5.2.1 Data selection

The size of the dataset influences the performance of the RS. The performance of the BMF algorithm decreases as the size of the training set gets larger as seen in the previous chapter. Based on the results of time window truncation in Chapter 4, we limit the number of ratings in the training set to 6 million ratings. This corresponds to a window size of 6 in the previous experiments. Next, we detail the process we employ to select the training window.

The analysis of the temporally ordered ML-10M dataset in Section 3.3 shows that the number of users, items, and ratings ramp up sharply at the beginning. Figure 3.8 shows the number of inactive users who have quit the system on a given date i.e. the date of their last rating. The number of inactive users is very high before the year 1999. The beginning portion of the ML-10M dataset will considerably increase the number of users and items that are not part of the test set. In order to reduce the number of users and items that are not evaluated in the training set, we discard the ratings which were recorded before the year 1999. This is illustrated in Figure 5.1.

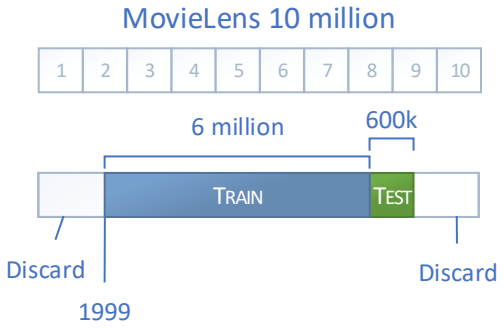


Figure 5.1: Selection of training and test ratings matrix from the ML-10M dataset.

We use the hold-out test strategy for evaluation. Since the ratings are temporally ordered, we select the data immediately following the training set as the test set. The number of ratings in the test set is limited to 10% of the ratings in the training data. The selected data window of training and test sets contain 6.6 million ratings of which 6 million ratings constitute the training set Tr_{full} and the remaining 600,000 ratings constitute the test set Te. We designate this data window as the MovieLens 6.6 million ratings (ML-6M) dataset and is used in the rest of the thesis. The number of users and items in this data window is 36,804 and 9,132 respectively. The test set contains 33,806 users and 8,857 items. The overall density of the selected data window is 1.96%. The summary statistics of the ML-6M dataset are in Table 3.1.

Table 5.1: Summary statistics of user profile truncated training sets.

Truncate Length	Train Items	Train Ratings	Density
10	7,438	337,617	0.13%
20	7,954	674,367	0.25%
30	8,193	983,537	0.36%
50	8,444	1,505,573	0.53%
100	8,646	2,480,618	0.85%
200	8,739	3,687,324	1.25%
300	8,784	4,409,973	1.49%
400	8,803	4,875,311	1.64%
500	8,818	5,184,757	1.74%
750	8,834	5,607,516	1.88%
1000	8,851	5,798,825	1.94%
Full	8,857	6,000,000	2.00%

5.2.2 Hyperparameter optimization

The experiments in this chapter use the MovieLens 6.6 million ratings (ML-6M) dataset. The number of factors, *nf*, is set as 20 as discussed in Section 4.2.1. The learning rate, *lr*, is optimized by a grid search for $lr \in [0.001, 0.002, 0.0025, 0.005, 0.0075, 0.01, 0.02]$. The performance is measured after 100 iterations. The results of the hyperparameter optimization for the training data is shown in Figure 5.2. Based on the observed RMSE values, the BMF algorithm for the given training dataset is optimized at $lr = 0.002$ and $nf = 20$. The

baseline RMSE for the MovieLens 6.6 million ratings (ML-6M) dataset is 0.898. The above hyperparameters are used in experiments hence forth.

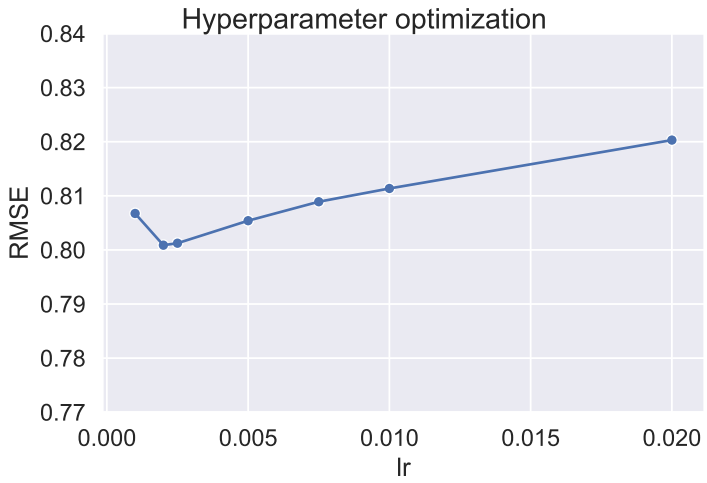


Figure 5.2: Hyperparameter optimization on learning rate (lr) for the training set (6 million ratings) of the MovieLens 10 million dataset.

5.2.3 Experimental setup

The profile truncation experiments evaluate the performance of BMF algorithm for truncated training sets $Tr_{truncated}$ for varying user profile lengths. The truncated training sets $Tr_{truncated}$ are generated using the Algorithm 3 for the following profile truncation lengths: $truncate\_len \in [10, 20, 30, 50, 100, 200, 300, 400, 500, 750, 1000]$. The summary statistics of the generated data for profile lengths is listed in Table 5.1. We observe that the number of ratings and the density of the ratings matrix decreases as the truncate length decreases. The BMF algorithms is used to evaluate the different truncated training sets against the constant test set, Te of the ML-6M dataset.

To understand the impact of the cold-start users on the performance of the BMF algorithm, the trained models are evaluated against test sets of differing cold-start percent of users–0%, 25%, 50%, 75%, 100%. In a test set with 0% of cold-start users, all the users in the test set are part of the training set. Whereas in a test set with 100% of cold-start users, all the users in the test set are not contained in the training set. The users in remaining cold-start levels are cumulatively selected from a randomly ordered list of cold-start users. Including the full test set Te, six different test sets are generated based on the above criteria. The BMF algorithm is evaluated against each of these test sets for all of the truncated training sets.

5.3 Results

The overall RMSE observed for the entire test set against different training sets of varying user profile lengths is shown in Figure 5.3. The performance of the BMF algorithm is not impacted when the profile length is greater than 500. Then the RMSE values increase slightly with a decrease in the profile size for up to $truncate\_len = 200$. The RMSE increases rapidly as the length of the profile decreases. Figure 5.4 shows the performance evaluation of the BMF algorithm in the presence of different levels of cold-start users.

For the test set with 0% cold-start users, the performance impact flattens after a profile length of 300 ratings. In the case of cold-start only users, the RMSE error is high when compared to overall performance. The presence of cold-start users increases the RMSE

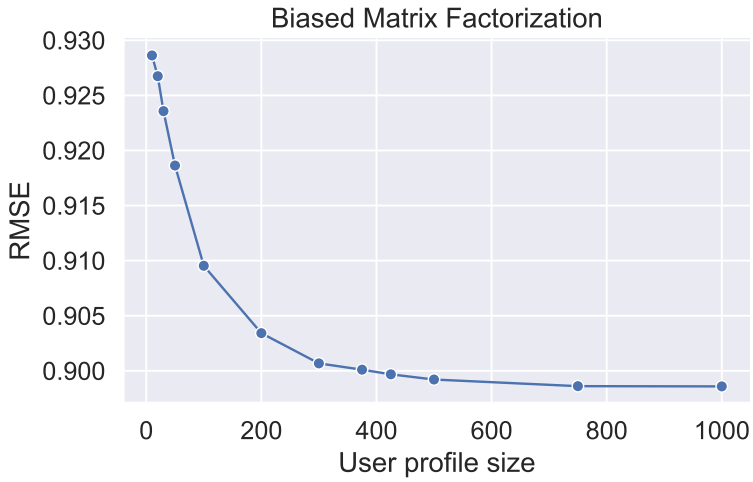


Figure 5.3: Impact of length of user's profile in terms of RMSE for the entire test set.

dramatically irrespective of the length of the user's profile. The RMSE errors for all the test sets with cold-start users follow the trend of 0% cold-start users.

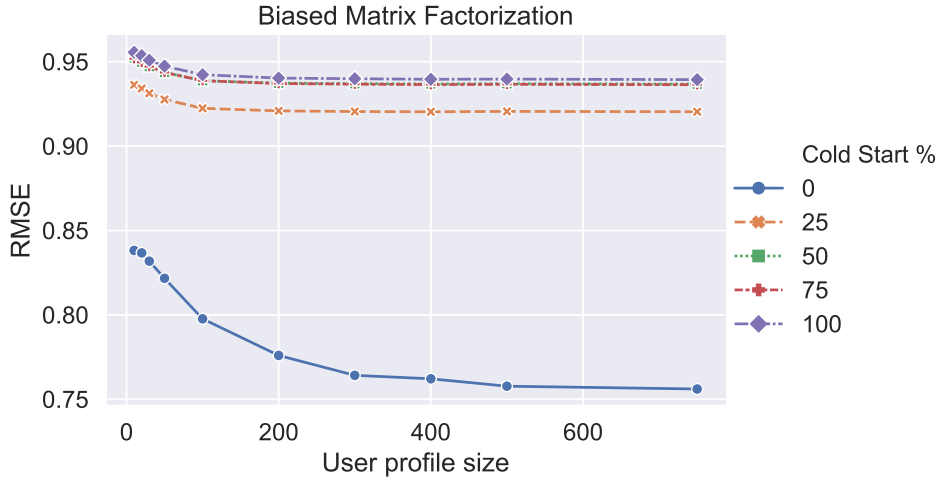


Figure 5.4: Impact of length of user's profile for select percentage of cold-start users.

5.4 Discussion

Profile length truncation reduces the length of all the user profiles resulting in a training set with profiles lengths less than the desired length. We can concur that the overall performance of BMF algorithm increases for shorter profiles and does not increase when the profile size is beyond a certain point. Moreover, by isolating the performance of the BMF algorithm without the cold-start users, we can demarcate the impact of user profile truncation. From Figure 5.3, and Figure 5.4 we determine that the overall performance flattens after user profile length of 300 latest ratings. Beyond this, the addition of older ratings will not increase the accuracy of the RS. Truncating the user profiles shorter than 300 will adversely affect the performance of the BMF. It shows a gulf in the performance levels of cold-start and trained users. On the other hand, using longer profiles does not improve the accuracy any

further. This emphasizes the need for profiles with a defined maximum number of ratings for effective recommendation.

The value of the longer profiles that are truncated is not analyzed. Amatriain et al. [4] show that the smaller set of expert users have more impact on the outcome of the RS. But, these expert users are more active and tend to have longer profiles. When we truncate the longer profiles, we inadvertently impact the wisdom imparted by these expert users. The value of the longer profiles and its impact warrants further study. This raises the question of whether we can truncate user-profiles selectively to decrease the impact of active users. As a follow up to this, we study the impact truncating a select percentage of the users in Chapter 6.

We are not considering the item distribution when truncating profiles. The user profile truncation may eliminate some items from the $Tr_{truncated}$ set. For instance, items that are rated in the past with less number of ratings are more susceptible. From Table B.1, we observe the number of items in the training set decreases as the length of the truncated profile becomes smaller. Around 1% of the items are lost when we truncate to a length of 200. Items are lost rapidly as we truncate the user profile to smaller lengths. Such items could become cold-start items when it is rated exclusively by users with long profiles who are truncated. This prompts further research to study the impact on the items when user profiles are truncated.

The number of users in the training set is the same after truncation. As a result, the number of cold-start users remains the same as well. The newer cold-start users decrease the prediction accuracy irrespective of the length of the user profile. From Figure 5.4, we can concur that the errors induced by the addition of cold-start users diminish as more cold-start users are added. addition of new cold-start users diminishes In a real-world scenario, the increase of cold-start users is unpredictable. The popularity of online platforms is determined by the number of new users. When a platform gets more popular, a large number of users join the system which could rapidly increase the number of cold-start users and in such cases, the rapid increase in cold-start users is a desirable trait. Hence in a RS with a high enough number of cold-start users, the new cold-start users are less likely to impact the overall prediction accuracy.

Truncation of the user profile reduces the amount of user data stored in the system. This reduced footprint could potentially limit de-anonymization attacks that rely on obscure item ratings i.e. items that are rarely rated. This method does not eliminate such attacks and it does not cover smaller profiles that are not truncated. The smaller profiles, on the other hand, could lead to malicious activity in the system. An astute user who knows the data minimization policy could inject several malicious rating [36] into the system which will affect the overall performance of the RS.

Not all of the user's data contribute equally to the outcome of the RS [13]. Some noisy data might be present in the ratings. Though truncating the historical ratings enhances the recency effects in the dataset and the overall performance of the RS, this may limit other user experience traits such as novelty and serendipity. Also, the ratings by the same user could be different when rated at two different points in time. For instance, the ratings by the same user as a child and an adult might vary considerably. Though the taste of the user changes over time, the user may still prefer items based on prior tastes. Truncating historical data could alter the overall profile characteristics of this user and predictions for other users. This raises the question of whether we can preserve the historical tastes of the users. As a follow up to this, we study the splitting up of a long user profile into smaller pseudo-user profiles in Chapter 7.

Chapter 6

Selective user profile truncation

This chapter advances the work done in the previous chapter by proposing the third data minimization technique to answer the third research question by truncating the user profiles of select users. First, we introduce the third research question, followed by the methods used to answer them in 6.1. Then we brief the experiments in Section 6.2, followed by the results in Section 6.3. Finally, we discuss the findings in Section 6.4.

The third data minimization technique addresses the following research question:

RQ3: *How many users can truncate their profiles without affecting the overall performance of the system (on average over all test users)?*

6.1 Methodology

The third data minimization technique, selective user profile truncation, extends the previous minimization technique. One of the selection criteria for truncating a user profile is its size. Similar to the second technique, we set the size of the long users' profiles. The profiles that are longer are selected and then the second criterion is applied.

The second criterion for selection is the percentage of users. Unlike the previous technique where all the users were truncated, only a selected % of the longer user profiles are truncated. % of the users are selected based on their time of the last rating. Long users who are most inactive are truncated first, and the recent users are truncated last. The methodology to answer the above research question is detailed in the following section.

6.1.1 Selective user profile truncation

The training data Tr_{full} is truncated based on two variables–truncation length $truncate\_len$ and selected users $truncate\_percent$. The truncation length sets the maximum profile length of the users in the training set and is similar to the methods employed to answer the previous research question in Chapter 5.

The second variable, $truncate\_percent$, selects the users $U\_truncate$ for whom the user profile truncation is applied. The user selection for profile truncation applies only to users whose profiles are longer than the truncation length of $trunacte\_len$. Profiles shorter than $truncate\_len$, are added to the training set $Tr_{trucated}$ without truncation. The time ordering criterion is used for profile truncation i.e. profiles with the oldest rating are selected first for truncation. This way, more recent profiles are truncated when the percentage of users to be truncated is higher. For the users in $U\_truncate$, a percentage of the users determined by $truncate\_percent$ is selected for truncation. The specific steps used to truncate the user profiles and generate the training data sets are defined in the function $truncate\_percent\_user\_profile$ in Algorithm 4.

Algorithm 4: Truncate the profile size of a percentage of the users in the training set whose profiles are longer than the truncation length.

```
1  function truncate_percent_user_profile (Tr_full, truncate_len, truncate_percent)
     Input  : Temporally ordered, full training set Tr_full
              Maximum length of a user's profile after truncation truncate_len
              Percentage of longer profiles to be truncated truncate_percent
     Output : Truncated Training Set(Tr_trucated)
2  Tr_trucated ← ∅
3  U_unique ← {user}, ∀user ∈ Tr_full
4  U_long ← get_longer_profiles(Tr_full, truncate_len)
5  n_users ← length(U_long) * truncate_percent/100
6  foreach user ∈ U_unique do
7      if user ∈ U_long then
8          n ← 0
9          while n <= n_users do
10             n ← n + 1
11             Profile ← Tr_full[user]
12             end ← length(Profile)
13             start ← end − truncate_len
14             Profile_trucated ← [Profile_i]_{i=start}^{end} ;      // truncate the profile
15             Tr_trucated ← Tr_trucated||Profile_trucated
16     else
17         Tr_trucated ← Tr_trucated||Profile
18 return Tr_trucated
```

6.2 Experiment

This experiment uses the ML-6M dataset as the base dataset. The first 6 million ratings constitute the full training set Tr_{full}. The remaining 0.6 million ratings constitute the test set Te. The selective user profile truncation method is applied on Tr_{full} using the function $truncate\_percent\_user\_profile$ defined in Algorithm 4 for the following parameter combination:

$truncate\_pecent \in [20\%, 40\%, 60\%, 80\%, 100\%]$

$truncate\_len \in [10, 20, 30, 50, 100, 200, 300, 400, 500]$

In all, 90 truncated training sets are generated. The summary statistics of the generated training sets are in Table B.1. The performance of the selective user profile truncation method evaluated using BMF for all the generated truncated training sets. The evaluation uses three distinct test sets to determine the overall performance, cold-start only performance, and zero cold-start performance. The hyperparameter optimization from Section 5.2.2 is used here. The optimized values for hyperparameter $lr = 0.002$ and $nf = 20$ are used in this experiment.

6.3 Results

The overall performance of the BMF algorithm for the generated training sets is shown in Figure 6.1. As the percentage of the selected users for truncation increases, there is an increase in RMSE as well. This effect is more pronounced for truncations which result in profile sizes $truncate\_len <= 100$. For the remaining longer profile sizes, the impact is minimal. The performance of the selective user profile truncation for users who are part of the training set and the cold-start users are shown in Figure 6.3 and Figure 6.2 respectively.

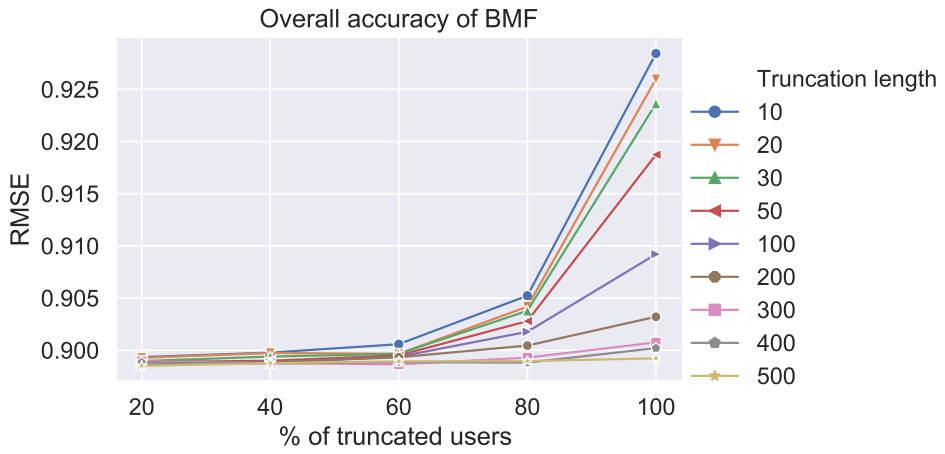


Figure 6.1: Overall performance impact of user profile truncation on a select percentage of users.

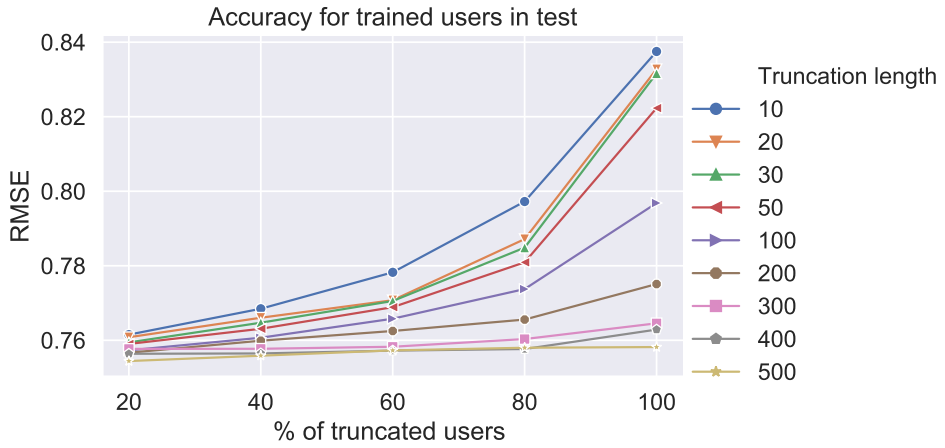


Figure 6.2: Performance impact of user profile truncation on a select percentage of users for previously seen users.

6.4 Discussion

In the selective user profile truncation, both the size of the user profile and the percentage of users to be truncated have an impact on the performance of the RS. The results show that the accuracy of the BMF algorithm is not affected when the percentage of people to be truncated is less than 40%. The smaller truncation lengths start to show higher error rates when 60% of the users are truncated. Performance decreases slightly at 60% when the truncation length is less than 50. For truncations affecting more than 60% of the users, the accuracy suffers appreciably with decreases in truncation length as seen in Figure 6.1. The error rates are minimal for profile lengths of 300 and more. This confirms the results from Chapter 5 that user profile truncation on long users yields better performance.

The findings show a similar pattern when using a test set with zero cold-start users as seen in Figure 6.2. The users in ML-10M have a minimum profile length of 20 ratings. From Figure 6.2, we observe that for smaller profile lengths of size 20, 30, and 50, the RMSE performance difference is minimal.

We expect the errors of cold-start only users to increase while truncating users. When

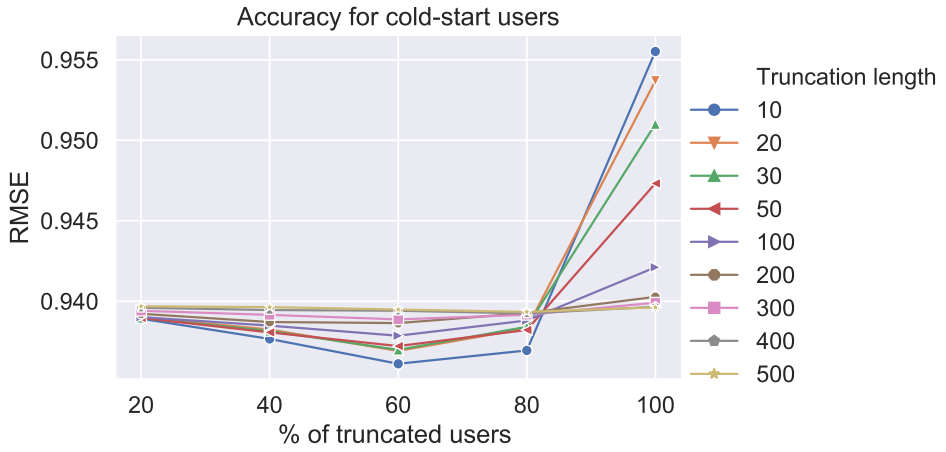


Figure 6.3: Performance impact of user profile truncation on a select percentage of users for cold-start users.

we truncate all the users i.e. with truncation percentage of 100%, the error for cold-start only users is high and is on par with the results from the previous experiment in Chapter 5. The prediction accuracy for cold-start only users for other truncation percentages was unexpected as the cold-start RMSE gradually decreases for truncated users with short truncation lengths as shown in Figure 6.3. For the truncation percentage of 60%, we observe that the smaller the truncation length, the greater is the decrease in RMSE. This effect could be similar to [4] where expert users impart wisdom to others. In our experiment, the long user are akin to the expert user in [4]. This shows that truncating users selectively up to a certain % of users does not damage the performance of the system and may slightly help. Beyond this threshold percentage of users, the performance decreases. This can be observed at the truncate percent of 80% and 100%.

Unlike the truncation in the previous experiment in Section 5.4, the selective user profile truncation preserves the value of the long user profiles. Based on the findings, we conclude that up to 60% percentage of users can selectively truncate their profile history to consist of 20 ratings i.e. have a minimal amount of ratings without compromising the effectiveness of the system.

The results obtained for is in line with user-controlled filtering by Wen et al. in [47]. The main difference between the two lies in the criteria used for selecting the users to be truncated. While we use the number of ratings for the user, $N_{ratings}$, the authors in [47] chose the number of days, N_{days}. The selection criteria might differ based on the domain. For instance, in the case of news recommenders where the timeliness of data plays a significant role in the prediction outcome [48], using the past N days of ratings might be more relevant. Based on the analysis of ML-10M dataset in Section 3.3, we argue that user profile truncation based on a set duration of N_{days} will inevitably increase the number of cold-start users. Hence the user profile truncation in [47] could result in an increasing number of cold-start users over time as users become inactive. This could severely affect the users who sparingly use the system. Keeping a fixed number of ratings in the user profile, on the other hand, will reduce the cold-start problem induced by truncation in [47].

Though the selective user profile truncation performs better than user profile truncation in the previous chapter, it suffers from similar drawbacks. By selective user profile truncation, this methods reduces some of the drawbacks of plain user profile truncation. Though susceptible to injection attacks, it can withstand better when compared with plain user profile truncation because of the presence of long user profiles. As with plain user truncation, some noisy data may be present in this case and will have an influence on the prediction.

Chapter 7

Long user-profile splits

This chapter addresses the fourth research question, RQ4, using the final data minimization technique proposed in this chapter. First, we define the terms and provide an overview. Section 7.1 details the methodology used to answer the research question. Section 7.2 details the design of the experiments, followed by the results in Section 7.3. Finally, Section 7.4 discusses the findings.

The fourth data minimization technique addresses the following research question:
RQ4: *If long user profiles are split instead of truncated, does this help the overall performance of the system (on average over all test users)?*

We hypothesize that a long enough user profile could be split into more than one pseudo-user profiles. We want to study the impact of the user profile splitting on the overall performance of the RS. The following section details the methods used to answer the above research question.

7.1 Methodology

The fourth and the final data minimization technique in this thesis does not reduce the number of ratings, unlike the previous techniques that reduced the number of ratings in training set by dropping user data. Instead, it aims to minimize the profile lengths splitting them and thereby increasing the number of users in the system. These artificially segmented users are called pseudo-users. They are part of the actual long user. Pseudo-users of a long user are non-overlapping.

The splitting strategy must discover meaningful user splits. [5] used a time-based strategy and split the users based on the context of the time, such as morning, evening, weekend, etc. We split the long users using two splitting strategies. The first one is a naive strategy where we control the length of the pseudo-user profiles. The second one uses the context of time. The activity gap denotes the time duration between two ratings by the user. We use activity gaps as a measure to create pseudo profiles. The rationale behind this approach is that we can accommodate the user's change in preferences over long periods of inactivity into pseudo-user profiles. We use two strategies to generate the training data–naive and activity-based, which are detailed in the following sections.

7.1.1 Naive truncation strategy

The naive strategy creates pseudo-users of the same profile length, $new\_profile\_len$. The long profiles are divided into as many pseudo-users as possible. The function *naive_virtual _profiles* used to generate the pseudo-users is defined in Algorithm 5. The short users below a threshold of $long\_len$ are appended to the resulting $Tr_{augmented}$ set directly. For each of the long users, the user profile is temporally split from the latest to the oldest ratings based on the desired length of the profile set via the parameter $new\_profile\_len$. The temporally oldest pseudo-user profile may contain less number of ratings than the desired profile length.

7.1.2 Activity based truncation strategy

Algorithm 6 defines the function used to generate the pseudo-user profiles based on gaps in user activity. Similar to the first strategy, this function appends the short profiles directly the training set $Tr_{augmented}$. For a long profile, the activity gaps in the user profile are considered for splitting the profile. A minimum activity gap of eight weeks is used along with the minimum length of the pseudo-user profile to determine the split point. By setting the minimum length of the pseudo-user profile, we avoid splitting the long profile into many smaller profiles. Unlike the naive strategy, this strategy generates pseudo-user profiles of varying sizes. All the pseudo-user profiles except for the temporally old profile are longer than $min\_profile\_len$.

Algorithm 5: Split long user profiles into pseudo-user profiles using a naive approach.

```
1  function naive_virtual_profiles (Tr_full, long_len, new_profile_len)
   Input  : Temporally ordered, full training set Tr_full
            Minimum length of a profile to be marked as long profiles long_len
            Maximum length of new user's profile after splitting new_profile_len
   Output : augmented Training Set(Tr_augmented)
2  Tr_augmented ← ∅
3  U_unique ← {user}, ∀user ∈ Tr_full
4  U_long ← get_longer_profiles(Tr_full, long_len)
5  new_id ← max(U_unique) + 1
6  foreach user ∈ U_unique do
7      Profile ← Tr_full[user]
8      if user ∈ U_long then
9          end ← length(Profile)
10         do
11             start ← end − new_profile_len
12             Profile_new_id ← [Profile_i]_{i=start}^{end} ;      // extract the profile
13             Tr_augmented ← Tr_augmented || Profile_new_id
14             new_id ← new_id + 1
15             end ← end − new_profile_len
16         while end > 0;
17     else
18         Tr_augmented ← Tr_augmented || Profile
19 return Tr_augmented
```

7.2 Experiment

The training data sets used in this experiment are generated from the ML-6M dataset. Like in previous experiments, the first 6 million ratings form the base training set Tr_{full} and the remaining 0.6 million ratings form the test set Te. The threshold to determine the long profiles is set by the parameter $long\_len$. We use $long\_len = 1000$ for the rest of the experiments. It results in 520 long users. Two methods are used to generate the training data sets. First, the function *naive_virtual_profiles* defined in Algorithm 5 is used to generate the data sets for $new\_profile\_len \in [100, 200, 300, 400, 500]$. Similarly, the function *new_pseudo_profiles* is used to generate the training data for the second method with different values of minimum profile length, $min\_profile\_len \in [100, 200, 300, 400, 500]$.

We adopt two different strategies for evaluation. First, we use the full test set Te consisting of 0.6 million ratings to get the performance over all the users. To study the

Algorithm 6: Split long user profiles into pseudo-user profiles using gaps in activity.

function new_pseudo_profiles (Tr_{full}, $long\_len$, $min\_profile\_len$)
Input : Temporally ordered, full training set Tr_{full}
Minimum length of a profile to be marked as long profiles $long\_len$
Minimum length of new user's profile after splitting $min\_profile\_len$
Output : $augmented\ \ Training\ \ Set(Tr_{augmented})$
$Tr_{augmented} \leftarrow \varnothing$
$U_{unique} \leftarrow \{user\}, \forall user \in Tr_{full}$
$U_{long} \leftarrow$ get_longer_profiles(Tr_{full}, $long\_len$)
foreach $user \in U_{unique}$ **do**
  if $user \notin U_{long}$ **then**
    $Tr_{augmented} \leftarrow Tr_{augmented} || Tr_{full}[user]$; `// append short profiles`
$new\_id \leftarrow max(U_{unique}) + 1$
$Tr_{pseudo} \leftarrow \varnothing$
foreach $user \in U_{long}$ **do**
  $Profile \leftarrow Tr_{full}[user]$ $pseudo \leftarrow 1$
  $start \leftarrow end \leftarrow length(Profile)$
  while $(start > 0)$ **do**
    $l \leftarrow end - start$
    $gap \leftarrow timestamp(Profile_{i=l}) - timestamp(Profile_{i=l-1})$
    if $(l > min\_profile\_len)\ \ \&\ \ (gap > 8\ weeks)$ **then**
      $Profile_{new\_id} \leftarrow [Profile_i]_{i=start}^{end}$; `// extract the profile`
      $Tr_{pseudo} \leftarrow Tr_{pseudo} || Profile_{new\_id}$
      $new\_id \leftarrow new\_id + 1$
      $end \leftarrow start$
    $start \leftarrow start - 1$
  if $end > 0$ **then**
    $Profile_{new\_id} \leftarrow [Profile_i]_{i=0}^{end}$; `// the oldest pseudo-profile`
    $Tr_{pseudo} \leftarrow Tr_{pseudo} || Profile_{new\_id}$
    $new\_id \leftarrow new\_id + 1$
$Tr_{augmented} \leftarrow Tr_{augmented} || Tr_{pseudo}$
return $Tr_{augmented}$

effect of user splitting, we exclusively test the newly created pseudo-users. The test ratings from the full test set belonging to the long users are replicated to each of the generated pseudo-users. For instance, if a long user is split into four virtual users, then all the test ratings of the long user are replicated for each of the four users. We use the BMF algorithm and measure the RMSE metric for each of the generated training datasets.

Table 7.1: Summary of new users created from 520 long users using naive and activity-based splitting strategies.

Min. profile length	Naive pseudo-users	Activity based pseudo-users
100	12,095	1,190
200	3,895	752
300	2,660	615
400	2,050	550
500	1,762	531

7.3 Results

The two splitting strategies generate a varying number of pseudo-users. The summary of new users created using the splitting strategies is in Table 7.1. The reported user counts include 520 long users. The number of pseudo-users created using the naive strategy is much higher than the activity-based strategy.

Table 7.2 shows the summary statistics of the generated augmented training data with pseudo-users using the naive strategy. The summary statistics for the activity-based strategy is in Table 7.3. The overall performance of the BMF algorithm is compared with the pseudo-user splitting strategies in Figure 7.1. The RMSE metric for both the user splitting strategies remains constant when the full training set is used. On the other hand, the RMSE of both the pseudo-user splitting strategies lowers as the minimum profile size increases.

Table 7.2: Summary statistics of naive pseudo-user splitting strategy.

Test strategy	Min. length	Ratings $\in$ *Te*	Users $\in$ *Te*	Users $\in$ *Tr*	Items $\in$ *Te*
Pseudo only	100	367,115	3,887	40,772	5,596
Pseudo only	200	190,121	2,017	37,181	5,596
Pseudo only	300	129,338	1,380	35,946	5,596
Pseudo only	400	100,590	1,066	35,336	5,596
Pseudo only	500	83,120	899	35,048	5,596
Full	100	600,000	5,602	40,772	8,139
Full	200	600,000	5,602	37,181	8,139
Full	300	600,000	5,602	35,946	8,139
Full	400	600,000	5,602	35,336	8,139
Full	500	600,000	5,602	35,048	8,139

Table 7.3: Summary statistics of activity-based pseudo-user splitting strategy.

Test strategy	Min. length	Ratings $\in$ *Te*	Users $\in$ *Te*	Users $\in$ *Tr*	Items $\in$ *Te*
Pseudo only	100	61,872	744	34,476	5,596
Pseudo only	200	37,739	436	34,038	5,596
Pseudo only	300	29,985	334	33,901	5,596
Pseudo only	400	24,851	284	33,836	5,596
Pseudo only	500	23,077	268	33,817	5,596
Full	100	600,000	5,602	34,476	8,139
Full	200	600,000	5,602	34,038	8,139
Full	300	600,000	5,602	33,901	8,139
Full	400	600,000	5,602	33,836	8,139
Full	500	600,000	5,602	33,817	8,139

7.4 Discussion

In this experiment, we study the effect of creating pseudo-user profiles from the long profiles using a naive and activity-based profile splitting strategies. The results show that the effect of user splitting has little impact on the performance of the RS when tested against the full test. However, on evaluating against a test set with pseudo only users, we observe that the performance increases as the minimum profile length increases for both the splitting strategies. In the case of naive splitting, the RMSE decreases linearly. In the case of activity-based splitting, only the minimum size of the profile is guaranteed and the actual

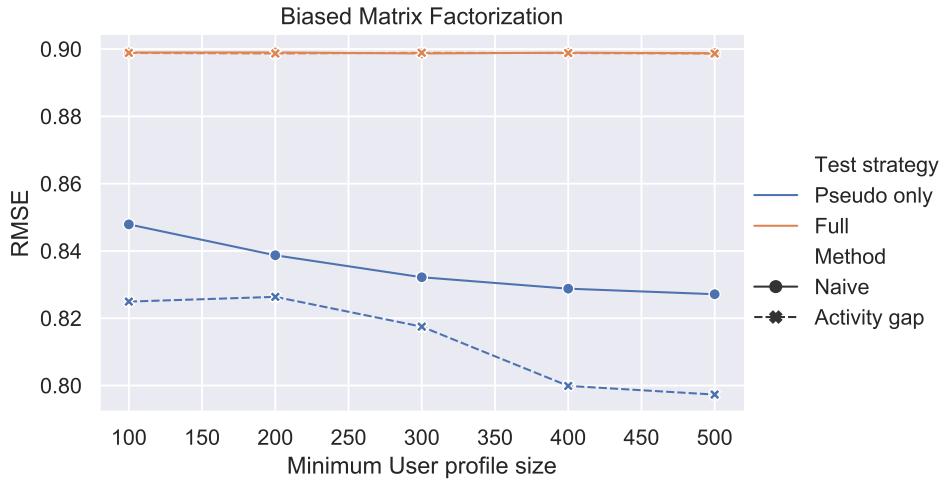


Figure 7.1: Performance impact of user profile truncation on a select percentage of users for cold-start users.

profile sizes are not. Though the number of pseudo-users in activity-based splitting is less, it performs better. This could be due to the longer size of the pseudo-user profiles. From the above observations in Table 7.1 and Figure 7.1, we concur that activity-based splitting strategy performs better than the naive splitting strategy in the context of evaluating the pseudo-users. This emphasizes our earlier findings in Chapter 6, and Chapter 7 that longer user profiles are needed for better performance.

We compare the observed performance of the RS over all users with the full ML-6M training set in Chapter 5. Though the number of users increases with user splitting, the number of ratings remains the same between the two. The number of items increases in the full test set when compared with the pseudo-user only test strategy. The prediction accuracy of the BMF remains the same for both the cases. Hence we conclude that the splitting of users does not adversely impact the performance of the RS.

Although the performance activity-based splitting is better than naive strategy, there exists scope for improvement in the selection of the split strategy for creating pseudo-users. Different split strategies such as clustering, user aggregation, and inclusion of contextual information could enhance the splitting of data. The impact of splitting the users using more split strategies warrants further study.

By splitting the user into pseudo uses, the temporal traits of the users are enhanced. The privacy implications of such a split are unknown at this time. By splitting the users, we not only reduce the number of ratings per user for long users but also create pseudo-users reducing the linkability of the user profiles. This could interfere with the de-anonymization attacks on large profiles. The potential of user splitting to reduce targeted de-anonymization attacks is a direction in which future work can be done.

The user splitting done here is the antithesis of how group recommendations work [7]. However, we think that the grouping of users could benefit from user splitting as a pre-processing step. Conversely, the grouping strategies could be modified for user splitting. In group recommenders, the presence of larger profiles work counter-intuitively as one of the primary strategies of group recommendation is to aggregate a larger common profile [7]. Reducing the larger profiles based on traits into pseudo-user profiles and then grouping them could yield better groupings. Also, [40] shows that grouping could help to improve privacy in recommender systems which could have implications for the work in our thesis. One direction for future work could be to study the impact of pseudo-users on the performance and privacy on applying group recommendation after the users are split.

Chapter 8

Conclusion and Future Work

In this chapter, we conclude the thesis by summarizing the results of our experiments in Section 8.1. Then we make suggestions for future research directions in Section 8.2.

8.1 Conclusion

In this thesis, we research about reducing the amount of data required to train a recommender system. We evaluate the Biased Matrix Factorization (BMF) algorithm using the MovieLens 10 million ratings (ML-10M) dataset. First, we conducted a through review of the literature to identify the challenges facing data minimization, followed by the analysis of the ML-10M dataset. To answer the main research question, we use four data minimization techniques. Here we list the research questions along with the conclusions we draw from the findings to address the main research question.

RQ1: *How does the temporal window truncation of training data affect the performance of the recommender system?*

We investigated the impact of temporal window based training data truncation. Here, we truncate data into 10 large temporal windows and evaluate the performance using the sliding window cross-validation method. The underlying characteristics of the training data changes as we change the length of the temporal window. The results show that the overall sliding window CV performance of the BMF algorithm increases initially for the window length of two, after which the RMSE increases. Furthermore, the results also show that the cold-start users have a considerable impact on determining the accuracy of the predictions. The fold-3 has the least number of cold-start users and as a consequence, better accuracy. The RMSE metric is different for different folds of data highlighting the timeliness effects. The accuracy of the system increases up to a window length of 3. Beyond this size, the performance of the system decreases slightly.

RQ2: *How does the truncation of users' historical rating profiles affect the performance of a recommender system?*

We analyze the impact of truncating the history of all of the users' profiles in Chapter 5 using the ML-6M dataset. The accuracy of the BMF algorithm increases as the profile size increases up to a length of 300 ratings per user when measured in terms of RMSE. After this, the inclusion of more ratings to the users does not improve the overall accuracy of the recommender system. This effect is less in the case of the cold-start users, where the accuracy of the BMF algorithm does not improve after a history length of 100. The findings illustrate that the user profile truncation on long users improves the accuracy of the system.

RQ3: *How many users can truncate their profiles without affecting the overall performance of the system (on average over all test users)?*

We proposed the third data minimization technique in Chapter 6, where a select percentage of the users' profiles are truncated. The findings show that truncating the smaller profiles while retaining long ones does not damage the performance of the system and may slightly help. It improves the accuracy of both trained and cold-start users. The presence of long user-profiles enables the truncation of users with short profiles. For the ML-6M dataset, we conclude that up to 60% of the users can truncate their profile to 20 ratings without compromising the accuracy of the recommender system. The results further emphasize the need for a certain number of long user profiles.

RQ4: *If long user profiles are split instead of truncated, does this help the overall performance of the system (on average over all test users)?*

The final data minimization technique proposed in Chapter 7 splits the long user profiles into smaller pseudo-profiles using naive, and activity-based split strategies. There is no noticeable difference in the accuracy when we compare the overall performance of the system with and without pseudo-users. When we isolate the pseudo-users and evaluate them separately, we find that the activity-based user splitting performs better than the naive user splitting. We conclude that splitting a long user into pseudo-users does not adversely affect the performance of the recommender system.

8.2 Future Work

In this thesis, we have demonstrated that reducing the amount of data required to train a recommender system could improve the performance. The methods proposed in Chapter 5, Chapter 6, and Chapter 7 opens new areas for research. Despite the findings from this work, there exist several issues that warrant further research. Many topics discussed in the above chapters remain unexplored. The suggestions for future work are discussed in this section.

- The impact of the combination of the different data minimization techniques is not studied. More experiments are required to analyze their combined impact. We leave the empirical study of these conditions as an important future work.
- We limited the scope of the experiments in this thesis to rating prediction. The use of the top-N recommendation task is becoming popular of late and is widely researched in the literature. We leave the work on data minimization in the top-N recommendation task as another significant area for future work. The proposed data minimization techniques would serve as a starting point to guide that research.
- The experiments in this thesis are limited to offline evaluation using RMSE metrics. We are unsure of how much the RMSE improvements translate to user-perceived improvements. The end-user does not necessarily observe the improvements in the accuracy metrics. The lack of online evaluation is a significant limitation in all our experiments. This limitation could be addressed in future work.
- In this thesis, the experiments only use the rating information for the recommendation task. A hybrid recommender using both the temporal and social information or contextual information could provide better prediction accuracy. One area for future research is to apply the proposed data minimization techniques and determine its validity using different types of recommender systems in other domains.
- Generalization – In this thesis, the experiments evaluate the BMF algorithm using ML-10M dataset. Due to this limited scope in the selection of dataset and algorithm, the results in this experiment might not hold for other datasets or algorithms. We must explore the characteristics of the other datasets that would support the hypothesis of this research. Also, evaluating other collaborative filtering algorithms using the same methodology is a direction for future work.

- Privacy – One advantage of data minimization is enhanced privacy as less data is retained and could reduce data de-anonymization attacks. At the same time, it could become vulnerable to attacks from malicious actors. One direction for future work would be to measure the privacy implications of the proposed data minimization techniques. Furthermore, the experiments in this thesis could be set up in conjunction with existing privacy-preserving RS models for measuring their combined effectiveness.

Bibliography

[1] ADOMAVICIUS, G., AND TUZHILIN, A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions, 2005.

[2] AGGARWAL, C. C. *Recommender Systems*. Springer International Publishing, Cham, 2016.

[3] AL-HADI, I. A. A. Q., SHAREF, N. M., SULAIMAN, M. N., AND MUSTAPHA, N. Review of the temporal recommendation system with matrix factorization. *International Journal of Innovative Computing, Information and Control 13*, 5 (2017), 1579–1594.

[4] AMATRIAIN, X., LATHIA, N., PUJOL, J. M., KWAK, H., AND OLIVER, N. The wisdom of the few: A collaborative filtering approach based on expert opinions from the web. In *Proceedings - 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009* (2009).

[5] BALTRUNAS, L., AND AMATRIAIN, X. Towards Time-Dependant Recommendation based on Implicit Feedback. In *Workshop on Context-aware Recommender Systems in conjunction with the 3rd ACM Conference on Recommender Systems* (2009).

[6] BENNETT, J., AND LANNING, S. The Netflix Prize. In *Proceedings of KDD Cup and Workshop* (2007).

[7] BORATTO, L., AND CARTA, S. State-of-the-Art in Group Recommendation and New Approaches for Automatic Identification of Groups. In *Studies in Computational Intelligence* (2010).

[8] CACHEDA, F., CARNEIRO, V., FERNÁNDEZ, D., AND FORMOSO, V. Comparison of collaborative filtering algorithms. *ACM Transactions on the Web* (2011).

[9] CAMPOS, P. G., BELLOGÍN, A., DÍEZ, F., AND CHAVARRIAGA, J. E. Simple time-biased KNN-based recommendations. In *Proceedings of the Workshop on Context-Aware Movie Recommendation* (2010).

[10] CAMPOS, P. G., DÍEZ, F., AND CANTADOR, I. Time-aware recommender systems: A comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction 24*, 1-2 (2014), 67–119.

[11] CAMPOS, P. G., DÍEZ, F., AND SÁNCHEZ-MONTAÑÉS, M. Towards a more realistic evaluation: Testing the ability to predict future tastes of matrix factorization-based recommenders. In *RecSys'11 - Proceedings of the 5th ACM Conference on Recommender Systems* (2011), pp. 309–312.

[12] ÇANO, E., AND MORISIO, M. Hybrid recommender systems: A systematic literature review. In *Intelligent Data Analysis* (2017), vol. 21, pp. 1487–1524.

[13] CHOW, R., JIN, H., KNIJNENBURG, B., AND SALDAMLI, G. Differential Data Analysis for Recommender Systems. In *Proceedings of the 7th ACM conference on Recommender systems - RecSys '13* (te, 2013).

[14] CUNHA, T., SOARES, C., AND DE CARVALHO, A. C. Selecting collaborative filtering algorithms using metalearning. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, T. Calders, F. Esposito, E. Hüllermeier, and R. Meo, Eds., no. September in Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg, jul 2016, pp. 393–409.

[15] DING, Y., AND LI, X. Time weight collaborative filtering. In *International Conference on Information and Knowledge Management, Proceedings* (2005).

[16] DING, Y., LI, X., AND ORLOWSKA, M. E. Recency-based collaborative filtering. In *Conferences in Research and Practice in Information Technology Series* (2006).

[17] EKSTRAND, M. D., RIEDL, J., AND KONSTAN, J. A. Collaborative filtering recommender systems. *Foundations and Trends in Human-Computer Interaction 4*, 2 (2010), 81–173.

[18] FRANKOWSKI, D., COSLEY, D., SEN, S., TERVEEN, L., AND RIEDL, J. You Are What You Say : Privacy Risks of Public Mentions. *Public Policy* (2006).

[19] GANTNER, Z., RENDLE, S., FREUDENTHALER, C., AND SCHMIDT-THIEME, L. MyMediaLite: A Free Recommender System Library. In *Proceedings of the 5th ACM Conference on Recommender Systems (RecSys 2011)* (2011).

[20] GORDEA, S., AND ZANKER, M. Time Filtering for Better Recommendations with Small and Sparse Rating Matrices. In *Web Information Systems Engineering – WISE 2007*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 171–183.

[21] GUNAWARDANA, A., AND SHANI, G. A survey of accuracy evaluation metrics of recommendation tasks. *Journal of Machine Learning Research* (2009).

[22] GUNAWARDANA, A., AND SHANI, G. Evaluating Recommender Systems. In *Recommender Systems Handbook*. Springer US, Boston, MA, 2015, pp. 265–308.

[23] HARPER, F. M., AND KONSTAN, J. A. The MovieLens Datasets. *ACM Transactions on Interactive Intelligent Systems 5*, 4 (2015), 1–19.

[24] HU, Q., YU, D., AND XIE, Z. Information-preserving hybrid data reduction based on fuzzy-rough techniques. *Pattern Recognition Letters* (2006).

[25] HU, Y., KOREN, Y., VOLINSKY, C., AND KOREN, Y. Collaborative Filtering for Implicit Feedback Datasets. In *Proceedings - IEEE International Conference on Data Mining, ICDM* (2008).

[26] KAGITA, V. R., PUJARI, A. K., AND PADMANABHAN, V. Virtual user approach for group recommender systems using precedence relations. *Information Sciences* (2015).

[27] KONSTAN, J., RIEDL, J., BORCHERS, A., AND HERLOCKER, J. Recommender systems: A grouplens perspective. *Recommender Systems: Papers from the 1998 Workshop (AAAI Technical Report WS-98-08)* (1998).

[28] KOREN, Y. Collaborative filtering with temporal dynamics. *Communications of the ACM 53*, 4 (apr 2010), 89.

[29] KOREN, Y., AND BELL, R. Advances in collaborative filtering. In *Recommender Systems Handbook, Second Edition*. Springer US, Boston, MA, 2015, pp. 77–118.

[30] KOREN, Y., BELL, R., AND VOLINSKY, C. Matrix factorization techniques for recommender systems. *Computer* (2009).

[31] LARSON, M., ZITO, A., LONI, B., AND CREMONESI, P. Towards Minimal Necessary Data: The Case for Analyzing Training Data Requirements of Recommender Algorithms. *FATREC Workshop on Responsible Recommendation* (2017).

[32] LEE, J., SUN, M., AND LEBANON, G. A Comparative Study of Collaborative Filtering Algorithms. *KDIR 2012 - Proceedings of the International Conference on Knowledge Discovery and Information Retrieval* (may 2012), 132–137.

[33] MIDDLETON, S. E., SHADBOLT, N. R., AND DE ROURE, D. C. Ontological user profiling in recommender systems. *ACM Transactions on Information Systems* (2004).

[34] MILLER, B. N., ALBERT, I., LAM, S. K., KONSTAN, J. A., AND RIEDL, J. MovieLens unplugged. *Proceedings of the 8th international conference on Intelligent user interfaces - IUI '03* (2003), 263.

[35] NARAYANAN, A., AND SHMATIKOV, V. Robust de-anonymization of large sparse datasets. In *Proceedings - IEEE Symposium on Security and Privacy* (2008).

[36] O'DONOVAN, J., AND SMYTH, B. Trust in recommender systems. In *International Conference on Intelligent User Interfaces, Proceedings IUI* (2005).

[37] RICCI, F., ROKACH, L., SHAPIRA, B., AND KANTOR, P. B., Eds. *Recommender Systems Handbook*. Springer US, 2011.

[38] RICCI, F., SHAPIRA, B., AND ROKACH, L. *Recommender systems handbook, Second edition*. 2015.

[39] SAID, A., AND BELLOGÍN, A. Comparative recommender system evaluation. *Proceedings of the 8th ACM Conference on Recommender systems - RecSys '14* (2014), 129–136.

[40] SHANG, S., HUI, Y., HUI, P., CUFF, P., AND KULKARNI, S. Beyond personalization and anonymity: Towards a group-based recommender system. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing* (2014), pp. 266–273.

[41] SHI, Y., LARSON, M., AND HANJALIC, A. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. In *ACM Computing Surveys* (2014).

[42] STECK, H. Evaluation of recommendations. In *Proceedings of the 7th ACM conference on Recommender systems - RecSys '13* (New York, New York, USA, 2013), ACM Press, pp. 213–220.

[43] SU, X., AND KHOSHGOFTAAR, T. M. A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence* (2009).

[44] THE EUROPEAN PARLIAMENT AND THE COUNCIL OF THE EUROPEAN. General Data Protection Regulation. https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679, 2016.

[45] VINAGRE, J. Time-aware collaborative fitering: a review. *In DOCTORAL SYMPOSIUM IN INFORMATICS ENGINEERING (p. 43) 2*, 5 (2013).

[46] VINCENT, N., HECHT, B., AND SEN, S. "Data Strikes": Evaluating the effectiveness of a new form of collective action against technology companies. In *The Web Conference 2019 - Proceedings of the World Wide Web Conference, WWW 2019* (2019).

[47] WEN, H., YANG, L., SOBOLEV, M., AND ESTRIN, D. Exploring recommendations under user-controlled data filtering. *Proceedings of the 12th ACM Conference on Recommender Systems - RecSys '18* (2018), 72–76.

[48] ZHANG, F., LIU, Q., AND ZENG, A. Timeliness in recommender systems. *Expert Systems with Applications* (2017).

[49] ZHANG, S., YAO, L., SUN, A., AND TAY, Y. Deep Learning based Recommender System: A Survey and New Perspectives. In *Engineering Letters* (2018), vol. 26, IEEE, pp. 203–209.

Appendix A

Biased Matrix Factorization

In this thesis, we used biased matrix factorization algorithm for experimentation. Here, we provide the steps to execute MyMediaLite's biased matrix factorization algorithm. Microsoft .NET runtime is required in case of Windows operating system. It can also run on Mono runtime with a minor penalty on the performance. We assume that training and test dataset is available as csv file. The csv file should contain *user*, *item*, *rating* tuples.

The main executable used for prediction is *item_predict.exe*. Some of the most important options set for execution are as follows:

```
mono item_predict.exe \
    --training-file    = "Training set" \
    --test-file        = "Test set" \
    --recommender      = "BiasedMatrixFactorization" \
    --prediction-file  = "Prediction result"
```

The following is an actual command for predicting a trained model for an experiment of the final data minimization technique, user splitting, in Chapter 7. It predicts the ratings for the test set `te.600k` on a pre-trained model trained with the `naive-100` training set.

```
key="naive-100"
te="te.600k"
pred="pred.naive-100.te600k"
mono /scratch/mkr/mml/bin/rating_prediction.exe \
    --training-file=/scratch/mkr/4/tr/$key \
    --load-model=/scratch/mkr/4/models/mod-$key \
    --load-user-mapping=/scratch/mkr/4/models/mu-$key \
    --load-item-mapping=/scratch/mkr/4/models/mi-$key \
    --test-file=/scratch/mkr/4/te/$te \
    --prediction-file=/scratch/mkr/4/pred/$pred
```

Appendix B

Selective user profile truncation

Table B.1: Summary statistics of the different training sets by truncation length and the % of the users truncated.

Truncate Length	% of users	Train Items	Train Ratings	Density %
500	100.0	8,818	5,184,757	1.74%
	80.0	8,824	5,331,832	1.79%
	60.0	8,829	5,482,985	1.84%
	40.0	8,851	5,657,023	1.89%
	20.0	8,854	5,830,716	1.95%
400	100.0	8,803	4,875,311	1.64%
	80.0	8,812	5,077,822	1.70%
	60.0	8,819	5,285,141	1.77%
	40.0	8,847	5,530,022	1.85%
	20.0	8,852	5,771,361	1.93%
300	100.0	8,784	4,409,973	1.49%
	80.0	8,798	4,701,014	1.58%
	60.0	8,811	5,004,472	1.68%
	40.0	8,845	5,344,941	1.79%
	20.0	8,851	5,676,048	1.90%
200	100.0	8,739	3,687,324	1.25%
	80.0	8,765	4,110,730	1.39%
	60.0	8,794	4,562,383	1.53%
	40.0	8,841	5,063,670	1.69%
	20.0	8,848	5,544,192	1.85%
100	100.0	8,646	2,480,618	0.85%
	80.0	8,711	3,131,665	1.06%
	60.0	8,792	3,863,966	1.30%
	40.0	8,838	4,587,775	1.54%
	20.0	8,845	5,306,055	1.77%
50	100.0	8,444	1,505,573	0.53%
	80.0	8,622	2,381,927	0.82%
	60.0	8,775	3,326,192	1.12%
	40.0	8,831	4,210,363	1.41%
	20.0	8,844	5,103,868	1.71%

Table B.1 Continued: Summary statistics of the different training sets by truncation length and the % of the users truncated.

	100.0	8,193	9,835,37	0.36%
	80.0	8,537	1,981,873	0.69%
30	60.0	8,753	3,026,293	1.02%
	40.0	8,825	3,986,276	1.34%
	20.0	8,842	4,974,470	1.66%
	100.0	7,954	6,743,67	0.25%
	80.0	8,460	1,736,456	0.61%
20	60.0	8,733	2,825,112	0.96%
	40.0	8,814	3,828,856	1.28%
	20.0	8,836	4,881,942	1.63%
	100.0	7,438	3,376,17	0.13%
	80.0	8,365	1,453,980	0.51%
10	60.0	8,713	2,602,494	0.88%
	40.0	8,809	3,677,183	1.23%
	20.0	8,836	4,807,346	1.61%