Temporal Performance Evaluation on Collaborative Filtering Algorithms

Miftahul Ridwan
STUDENT NUMBER: 2040778

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN DATA SCIENCE & SOCIETY
DEPARTMENT OF COGNITIVE SCIENCE & ARTIFICIAL INTELLIGENCE
SCHOOL OF HUMANITIES AND DIGITAL SCIENCES
TILBURG UNIVERSITY

Thesis committee:

Çiçek Güven, Ph.D.
Yash Satsangi, Ph.D.

Tilburg University
School of Humanities and Digital Sciences
Department of Cognitive Science & Artificial Intelligence
Tilburg, The Netherlands
May 2020

Preface

I want to express my deepest gratitude to Çiçek Güven, Ph.D. for her invaluable guidance, feedback, and support throughout the process of writing this work. I would also like to extend my gratitude to Yash Satsangi, Ph.D. for being part of the thesis committee for this work amid his busy schedule.

I would like to also thank Abigail Ho for her thought and support. I thank Andrie Zainal Zen and Andhika Candra Nughraha who always be there when I need.

This work is dedicated to my parents who, because their pray during the morning, the afternoon and the evening, enable me to proceed even in the most unlikely circumstances.

Temporal Performance Evaluation on Collaborative Filtering Algorithms

Miftahul Ridwan

In previous work on CF, researchers are mainly focused on comparing algorithms as well as proposing new and/or enhanced state-of-the-art algorithms claiming to outperform the existing ones. Experiment setting often started with splitting the data into training and test set, the algorithms are then trained and evaluated against the unseen test set. However, CF algorithms operate in a rather different setting once it is deployed in the production stage. As the number of user and the business grow over time, the algorithms should be trained periodically using all the available data to provide the user with up-to-date recommendation.

In this work, we evaluate CF algorithms on Netflix Prize dataset, by mimicking how they are deployed in production stage: incrementally updating the training and test set, as well as iteratively training and measuring the performances over the course of observation period. The algorithms are User-based k-NN, Item-based k-NN, SVD and SVD++ algorithms. By setting the update interval to monthly system update, we found that neither User-based k-NN nor Item-based k-NN algorithms' performances overlap with SVD and SVD++ algorithms. However, we found that SVD++ algorithm does not consistently outperformed SVD algorithm as previously suggested in the literature. Thus, our work indicate the urge to always perform temporal evaluation before claiming one algorithms to outperform the others.

1. Introduction

We rapidly produce and consume new information concurrently everyday, whether through social media post, e-commerce transaction, or our digital footprints such as reviews and browsing history. The accelerated production of information and more efficient distribution of that information through information and communication technology have known to caused information overload (Eppler and Mengis 2004). One of the essential mechanisms to fight information overload, according to Eppler and Mengis (2004), is through visualized, compressed and aggregated information. In practice, many modern businesses and applications provide a personalised list of products and services that their users might find relevant or useful (Gunawardana and Shani 2015). Such personalised list is generated by **R**ecommender **S**ystems (RSs). These systems ease a user's task of finding preferred and relevant products, as well as services, in the catalogue. They also help the user to narrow down the choices and reduce information overload.

Among many approaches, **C**ollaborative **F**iltering (CF) is the most extensively used approach used to design an RS (Thorat, Goudar, and Barve 2015). CF algorithms are giving recommendations by combining information from fundamentally different objects: users and items (Koren and Bell 2015). Moreover, Koren and Bell (2015) explain two primary approaches, which constitute the two main disciplines of CF: *the neighbourhood methods* and *latent factor models*. The neighbourhood methods use the similarities

between items, or alternatively between users, to predict user preferences for an item. On the other hand, the latent factor models approach the problem by expanding the bias baseline equation as discussed in Potter (2008) and taking into account user and item characteristics from latent factor space. Koren and Bell (2015) explains that latent factor space could be understood as the characteristics that define item or user: when the products are movies, factors might measure genre, actor or actress, or orientation to children; less well defined dimensions such as depth of character development or "quirkiness"; or completely unexplainable dimensions. Although it is not explicitly observed in the dataset, latent factor models inferred this user and item characteristics in the learning process.

In previous work on CF, researchers are mainly focused on comparing algorithms as well as proposing new and/or enhanced state-of-the-art algorithms claiming to outperform the existing ones. Experiment setting often started with splitting the data into training and test set, the algorithms are then trained using the training data and queried for recommendation. Performance is calculated by comparing the rating in the test set against the prediction. Then, it is reported in static, single numbers representation[1]. However, these algorithms are implemented in a different setting once it is deployed in the production stage. As the number of user and the business grow over time, the algorithms should be trained periodically using all the available data to provide the user with up-to-date recommendation. In Lathia (2010), the author finds that CF algorithms do not produce consistent output when evaluated over time, thus explicitly claiming one algorithm is better than the others based on static performance representations could be misleading, especially if the improvement in performance is fractional. Hence, Beel (2017) suggests that algorithm's performances over time are calculated and presented to better understand the magnitude of changes in algorithms' effectiveness.

In this work, we evaluate temporal performance of 4 CF algorithms to further investigate the claim made by Koren and Bell (2015) that latent factor models offer more accurate rating prediction than neighbourhood methods on Netflix prize dataset[2]. For the neighbourhood methods, we investigate the temporal performance of (1) User-based *k*-NN algorithm and (2) Item-based *k*-NN algorithm. For the latent factor models, we investigate the temporal performance of (3) **S**ingular **V**alue **D**ecomposition (SVD) algorithm, and (4) SVD with implicit feedback (SVD++) algorithm formulated by Koren and Bell (2015). We deploy temporal evaluation techniques proposed by Lathia (2010). Firstly, we determine the edge time ϵ: the cutoff time in the dataset for which the system will be trained upon for the first time. Deciding ϵ is crucial to alleviate system-wide cold-start problem, in which the system has to predict the rating for a user without any historical rating. We also decide our system update interval υ to monthly update $\upsilon = 30$-days. Performance evaluation is conducted by following the rule that algorithm's performance at time τ_t, based on all data available prior to τ_t, is calculated on test set of $\tau_t + \upsilon$ period, repeatedly until the last time point in the dataset. Thus, our research questions are formulated as follows:

RQ1: *To what extent the performance of CF algorithms varies when evaluated incrementally on Netflix Prize dataset over the course of observation period?*

1 Some of the most widely used performance representations are, among others: **R**oot **M**ean **S**quared **E**rror (RMSE), **M**ean **S**quared **E**rror (MSE), **M**ean **A**bsolute **E**rror (MAE) as well as precision and recall value from confusion matrix.

2 Available at https://www.kaggle.com/netflix-inc/netflix-prize-data.

RQ1a: *What is the range of prediction error for **User-based k-NN algorithm** when evaluated incrementally on Netflix Prize dataset over the course of observation period?*
RQ1b: *What is the range of prediction error for **Item-based k-NN algorithm** when evaluated incrementally on Netflix Prize dataset over the course of observation period?*
RQ1c: *What is the range of prediction error for **SVD algorithm** when evaluated incrementally on Netflix Prize dataset over the course of observation period?*
RQ1d: *What is the range of prediction error for **SVD++ algorithm** when evaluated incrementally on Netflix Prize dataset over the course of observation period?*

In this work, we acknowledge several limitations. The result in this work is dependent to the dataset we are using. Moreover, deciding which algorithm perform the best is beyond the scope of this work. We also understand that the algorithms we use in this work do not explicitly incorporate time information into the model (e.g., as time-weight decay). However, we are interesting to investigate how these algorithms that explicitly include time information will perform in temporal setting in the future research. Furthermore, we are not taking into account the feedback loop effect in RS in this work, where user actions are influenced by the output of RS and RS itself is trained based on user actions. This loop might degrade the quality of the recommendations because it is challenging to distinguish whether the user is watching a movie because they are enjoying it or because it repeatedly shown to them (Basilico and Raimond 2017).

Due to difference in sampling technique, system update interval, and hyper-parameter setting than those in Lathia (2010), we found no evidence that the performance of neighbourhood methods overlap with latent factor models on temporal setting. However, it is interesting to note that the lowest prediction error for neighbourhood methods is lower than the highest prediction error for latent factor models, although it is happening in different hyper-parameter setting and system update. We encourage future RS research to always perform temporal performance evaluation and present the result over the course of observation period. These results, not only provide a perspectives on how these algorithms perform in production stage, but also help the business to improve the design and the quality of their existing RS and provide better recommendation to the users. Indeed, better recommendation is essential reduce our exposure to information overload and, on a greater extent, improve our quality of life.

The remainder of the work is organized as follows: we discuss the development in RS literature surrounding the temporal evaluation protocol in Section 2; we explain the algorithms we examine in this work in Section 3; we explore the structure of the dataset and explain our evaluation procedure in Section 4; we then present the results in Section 5 and discuss them in Section 6. From these results, we conclude this work in Section 7 by providing the answer to our research questions and direction of future research.

2. Related Work

Temporal performance evaluation in this work revolves around how the algorithms perform over time. Beel (2017) explains that many RS researches have been incorporated time information, but rarely used in evaluation. Particularly in CF research, Campos, Díez, and Cantador (2014) discuss 3 common approaches of incorporating time information as contextual dimension, namely *continuous time-aware approach, categorical time-aware approach*, and *time adaptive approach*. Time information in continuous time-aware approach is modeled explicitly in the algorithms as a continuous variable. Example of this approach includes exponential time decay weight in Ding and Li (2005), which interpolates the time distance between reference time and target time under the assump-

tion that the more recent the rating, the more the information will be useful to predict the next rating, and time-aware SVD++ (timeSVD++) algorithm in Koren and Bell (2015), which uses bias terms such as user-specific, item-specific, and time-dependent bias to model user's interest drift over time.

In categorical time-aware approach, time information is incorporated as discrete contextual variables, including pre-filtering and contextual post-filtering techniques. Example algorithm of this approach includes user *micro-profiling* in Baltrunas and Amatriain (2009), which utilises time context, such as weekday or weekend, to give different recommendation, and support vector machine model for recommendation in Oku et al. (2006), which incorporates several contextual dimensions including time, company and weather into the algorithm. In addition, there are other techniques used in this approach, as documented by Jeunen (2019), such as time-aware **L**eave-**O**ne-**O**ut **C**ross-**V**alidation (LOOCV), incremental update technique, and sliding window evaluation technique.

Furthermore, time information is modeled through parameters and/or data dynamic relative to other features' dynamic over time in time adaptive approach. In a sense, feature information obtained at certain point in observation time, fed forward to the next algorithm iteration. Example algorithm of this approach includes Adaptive CF in Lathia (2010), which dynamically adjusted the number of neighbour k in kNN approach to minimize the prediction error for each system update, and Multiplicative Model in Karatzoglou (2011), which adds an item-factor vector that is consumed by specific user until certain period of time into the algorithm.

In this work, we follow time-aware evaluation method by incrementally updating the training and test set, as well as iteratively training and measuring the performances over the course of observation period. The result would be presented for every system update we do. To the best of our knowledge, there have been a *few* researches that present the result over some period of time and focused on either time-aware evaluation or temporal performance evaluation.

We are inspired by the work of Lathia (2010), who evaluate temporal performance using weekly update interval of 3 algorithms on MovieLens and Netflix prize dataset, namely (1) Bias Baseline model, (2) Item-based k-NN algorithm, and (3) SVD algorithm. The author reports both cumulative and non-cumulative prediction errors for 250 weekly system updates over the course of observation data. Using the number of neighbour $k = [20, 35, 50]$ for Item-based k-NN algorithm and $factor = 64$ for SVD algorithm, the author find that these algorithms' prediction errors overlap with each others, both in cumulative and non-cumulative prediction errors. Moreover, at its best performance, Bias Baseline model shows only fractional improvement than Item-based k-NN algorithm. Due to the fact that CF algorithms do not produce consistent prediction error when evaluated over time, the author signifies the difficulties when claiming one algorithm outperformed the others based on static, single number result. It is interesting to also note that the author highlights the performance of neighbourhood methods depend on the number of neighbour k being selected and the performance of SVD algorithm depends on the number of features (*factor*), the learning rate, the number of round (*epochs*) the algorithm are trained. Therefore, in this work, we are interested to investigate how CF algorithms perform in temporal setting using the same evaluation protocol when we set the hyper-parameter equally. We are using 5 values for both k and $factor$, that is $[10, 20, 30, 40, 50]$.

Time-aware evaluation could also be found in the work of Soto (2011), who evaluates 6 algorithms using monthly time-ordered test split on MovieLens 1M dataset, namely (1) User-based k-NN algorithm, (2) Time Decay and Time Truncation algo-

rithms, (3) Bias Baseline model, (4) Matrix Factorization algorithm, (5) Clustering models, and (6) AutoSimilarity algorithm. Instead of interatively training and measuring the algorithm over incrementally updated training set, Soto (2011) trains the algorithms only once, and evaluates them on 20 different time-ordered test sets. The author argues that by following this protocol, it makes cross-validation inside the same test data in the same test period possible. For instance, test period at time τ_t will have test data τ_{test} and by using this evaluation protocol, it is possible to perform 5-fold cross-validation on τ_{test}. The author reports the result for every test period. Furthermore, by using this evaluation protocol, the author finds that the change of test data distribution affect all algorithms almost equally: they are decreasing and increasing the prediction error of all algorithms on the same test period. Moreover, while the prediction error of several algorithms overlap with each others in non-cumulative result, it is not the case in cumulative result. Given the size of our dataset, we are not going to perform cross-validation. Instead, we are determine to use monthly update interval $\upsilon = 30$-days in our work. In spite of we are not going to perform cross-validation in this work, we are interested to investigate how CF algorithms will perform under this evaluation protocol in the future.

Furthermore, in the work of Jeunen, Verstrepen, and Goethals (2018), the authors evaluate 4 algorithms, namely (1) Popularity model, (2) User-based *k*-NN algorithm, (3) Item-based *k*-NN algorithm, and (4) SVD algorithm, using time-aware sliding window evaluation technique on *Retail* dataset: a proprietary dataset obtained from the logs of a live RS serving a Belgian retail website. The authors perform time-wise split that results in 30-fold of equal length data over the course of observation period: length in the sense of *n* number of days, equal length is in the sense of each fold contains the same number of days. From here, the authors perform sliding-window evaluation protocol: at time τ_t, the algorithms are trained using the data in $fold_t$ and evaluated on $fold_{t+1}$. The authors claim that this evaluation protocol is closely resemble A/B testing in online evaluation setting: where evaluation is based on user's live interaction with the system, instead of using historical data to measure algorithm's performance. The authors present the result in averaged recall values. Furthermore, by using this evaluation protocol, the authors finds that Item-based *k*-NN algorithm produces the highest recall value in all number of *k*. However, since the number of fold is arbitrary, its impact on algorithm's performance is remain unknown. Thus, we are not going to deploy this type of time-aware evaluation protocol in this work.

On the other work, Koren and Bell (2015) suggest that prediction accuracy of SVD algorithm is improved when taking implicit feedback into the equation, dubbed the equation as SVD++ algorithm. In this sense, implicit feedback is in the form of rental, purchase or browsing history. This historical data provide insight about user's preferences without necessarily the user provide the rating. Moreover, the authors explain that, for Netflix prize dataset, it does tell us about user's preferences by choosing to voice user's opinion and gave a rating (for instance, by assign "1" for rated item and "0" otherwise) along with the explicit rating data. The authors also report better performance for SVD++ algorithm compared to SVD algorithm, both algorithms show lower RMSE trends as number of factor increases. However, given the superiority over SVD algorithm, SVD++ algorithm is the least discussed algorithm in time-aware evaluation setting. This motivates us to include SVD++ algorithm into this work.

To summarise our work, we follow temporal evaluation technique discussed in Lathia (2010). However, we include 2 more algorithms which are not evaluated on the previous work: namely (1) User-based k-NN algorithm and (2) SVD++ algorithm. Since it is arbitrary, we change the update interval to monthly update to see how it is different

from weekly evaluation as previously done. Moreover, we use same hyper-parameter settings to evaluate the algorithms: we use $[10, 20, 30, 40, 50]$ as the number of k and $factor$, instead of $[20, 35, 50]$ for number of neighbour and 65 for number of factors in the previous work. Since the author does not mention about the details of learning rate and regularization parameter, we follow Koren and Bell (2015) to use $[lr = 0.005, reg = 0.02]$ for SVD and $[lr = 0.007, reg = 0.02]$ for SVD++ algorithm. For each factor, SVD and SVD++ are trained for 30 epochs. We further explain about the algorithms we evaluate in this work in the following Section 3.

3. Model

The previous section discussed methods to evaluate CF algorithms which taking into account the temporal structure in the dataset. In this section, we explain the algorithms we examine. As previously discussed, CF algorithms combine two fundamentally different information: users and items (Koren and Bell 2015). In doing so, Koren and Bell (2015) explain two primary approaches: the neighbourhood methods and latent factor models.

3.1 The Neighbourhood Methods

Neighbourhood methods are the most common approach used to develop CF algorithms (Koren and Bell 2015). In these methods, rating is estimated using known rating of either like-minded users on the same item (User-based *k*-NN algorithm) or the same user on similar items (Item-based *k*-NN algorithm), based on similarities measure, such as Pearson Correlation (Koren and Bell 2015). For instance, to approximate the similarities between user u_1 and u_2, Thakkar et al. (2019) formulate the equation 1 as follows:

$$sim(u_1, u_2) = \frac{\sum_{i \in I_{u1,u2}} (r_{u_1,i} - \bar{r}_{u_1})(r_{u_2,i} - \bar{r}_{u_2})}{\sqrt{\sum_{i \in I_{u1,u2}} (r_{u_1,i} - \bar{r}_{u_1})^2 \ \sum_{i \in I_{u1,u2}} (r_{u_2,i} - \bar{r}_{u_2})^2}} \tag{1}$$

In this equation, $I_{u1,u2}$ denotes the set of items rated by both u_1 and u_2, and $\bar{r}_u$ is the average rating $\bar{r}$ of the subsequent user u. Thus, we could predict the rating $\hat{r}$ for user u on item i using the equation 2 as follows:

$$\hat{r}_{u,i} = \bar{r}_u + \frac{\sum_{k \in K} sim(u, k) \times (r_{k,i} - \bar{r}_k)}{\sum_{k \in K} |sim(u, k)|} \tag{2}$$

In a sense, K denotes the set of k-Nearest Neighbours to user u who have been giving rating on item i. Using the same intuition, we could also estimate the item-item similarities using the equation 3 as follows:

$$sim(i_1, i_2) = \frac{\sum_{u \in U} (r_{u,i_1} - \bar{r}_{i_1})(r_{u,i_2} - \bar{r}_{i_2})}{\sqrt{\sum_{u \in U} (r_{u,i_1} - \bar{r}_{i_1})^2 \ \sum_{u \in U} (r_{u,i_2} - \bar{r}_{i_2})^2}} \tag{3}$$

In this equation, U denotes the set of users who rated both i_1 and i_2. $\bar{r}_i$ is the average rating $\hat{r}$ for subsequent item i. We could predict the rating $\bar{r}$ for user u on item i using the equation 4 as follows:

$$\hat{r}_{u,i} = \bar{r}_i + \frac{\sum_{n \in N} sim(i,n) \times (r_{i,n} - \bar{r}_n)}{\sum_{n \in N} |sim(i,n)|} \tag{4}$$

The intuition is relatively the same with predicting rating using User-based k-NN algorithm in equation 2. N denotes the set of n-Nearest Neighbours of item i who have been rated by user u. The advantage of neighbourhood methods as suggested by Koren and Bell (2015), among others, is the speed to which the algorithm adapt to new ratings. In a sense, because the algorithm estimated the similarities directly from the data without training process, new user would receive recommendation immediately after providing the first feedback.

3.2 Latent Factor Models

As suggested in Koren and Bell (2015), due to their superior accuracy over the neighbourhood methods, latent factor models are popular model RS researches. Latent factor models, such as SVD algorithm, perform reasonably well in retrieving information by decomposing latent semantic factors over sparse rating matrix. For instance, suppose that in latent factor space of dimensionality f, vector $q_i \in \mathbb{R}^f$ denotes the factor corresponds to item i and vector $p_u \in \mathbb{R}^f$ denotes the factor corresponds to user u. Koren and Bell (2015) suggest that the interaction $q_i^T p_u$ of user u over characteristic of an item i could be written as:

$$\forall\, q_i, p_u \in \mathbb{R}^f : q_i^T p_u = \sum_{k=1}^{f} q_{i_k} \cdot p_{u_k} \tag{5}$$

This interaction is then added to the pre-defined baseline model. The rating is then predicted by the following equation:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u \tag{6}$$

In equation 6, μ is denotes the system-wide average rating, whereas b_u and b_i are denotes the differences between system-wide average rating and user u and between system-wide average rating and item i respectively. Model parameters (b_i, b_u, q_i^T, p_u) are then learned using stochastic gradient descent of the following formula:

$$\min_{b*,q*,p*} \sum_{(u,i) \in \mathcal{K}} (r_{ui} - \mu - b_i + b_u - q_i^T p_u)^2 + \lambda(b_i^2 + b_u^2 + \|q_i\|^2 + \|p_i\|^2) \tag{7}$$

In equation 7, λ denotes regulatization to control overfitting. Koren and Bell (2015) dubbed this model as "SVD algorithm". Moreover, the authors also introduce a model that taking into account user's implicit feedback for even better accuracy. In this sense,

implicit feedback provides additional indication of user's preferences, even by only indicating which items users rate. For this operation, we use new vector $y_i \in \mathbb{R}^f$ for items i. This new vector would be used to characterize users based on their rating activity over set of items. The SVD algorithm with implicit feedback could be rewritten as follows:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T \left(p_u + \left| \frac{1}{\sqrt{R(u)}} \right| \sum_{j \in R(u)} y_j \right) \tag{8}$$

In equation 8, $R(u)$ denotes implicit feedback from user u on item i. Koren and Bell (2015) dubbed this algorithm as "SVD++" and suggests lower prediction error when evaluated on Netflix prize dataset, in compare to SVD model.

4. Experimental Setup

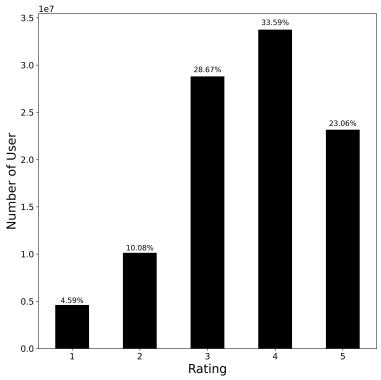
The task of RS takes known rating input in the data and predicts the values for a content that users have not rated, generates personalized lists of content that users might find relevant or useful. As we previously discussed, with the rating data and business grows, such system needs to be periodically trained to accommodate these changes. It is also trivial to understand that any changes in the rating distribution would affect the performance of any algorithm. In this work, we aim to investigate the extent to which algorithms' performances drift when iteratively and periodically trained to mimic temporal setting when these algorithms deployed in production stage. Understanding these changes over time would provide a perspectives on how these algorithms perform in production stage, thus help the business to improve the design of their existing RSs and provide better recommendation to the users. We evaluate temporal performance of 4 CF algorithms, namely (1) User-based *k*-NN algorithm, (2) Item-based *k*-NN algorithm, (3) SVD algorithm, and (4) SVD++ algorithm, using Netflix Prize dataset. We begin this chapter by exploring the characteristic of our dataset in Section 4.1, and further explaining out evaluation procedure in Section 4.2.

4.1 Data

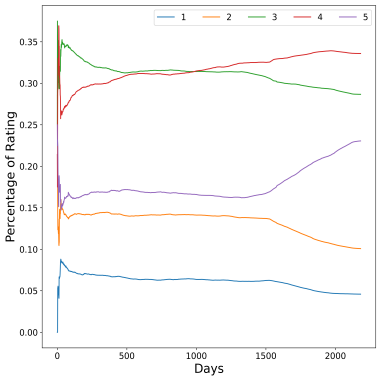
As previously mentioned, we use the Netflix Prize dataset to examine our algorithms' performances. Netflix prize was a rating prediction contest against Netflix-owned Cinematch algorithm with a grand prize of 1 million US dollar and was concluded in 2009. This dataset contains 480,188 unique users who rated 17,770 movies dated from 11 November 1999 until 31 December 2005. With more than 100 milion ratings data, the sparsity of this dataset is 98.82%. When we obtain this enormously sparse dataset, it comes in the form of 4 text files, with each file contains lines indicating a movie ID, followed by a colon, and in the next line is customer ID, rating, and date. The rating data itself is in the range of 1 to 5. We preprocess the data into flat table with 4 columns: $[\text{"}user\_id\text{"}, \text{"}date\text{"}, \text{"}rating\text{"}, \text{"}movie\_id\text{"}]$.

Furthermore, we investigate the rating structure, its mean and standard deviation, as well as each rating's proportion to the global rating on a daily basis.as shown in Figure 1. We also explore its temporal structure in terms of number of rating per day of the week. As shown in Figure 1a, the majority of users were given rating 4 to movies in the dataset. However, it is not always the case through out the observation period.

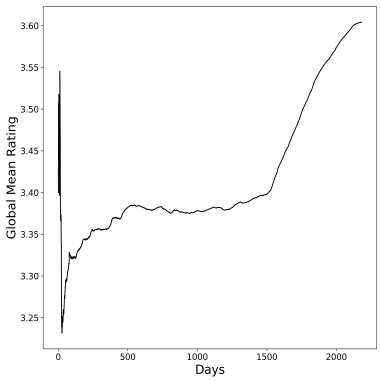
Figure 1: Rating Overview



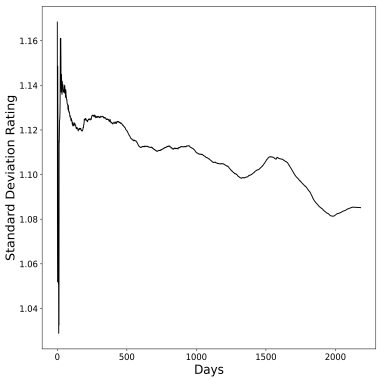
(a) Global Rating Distribution



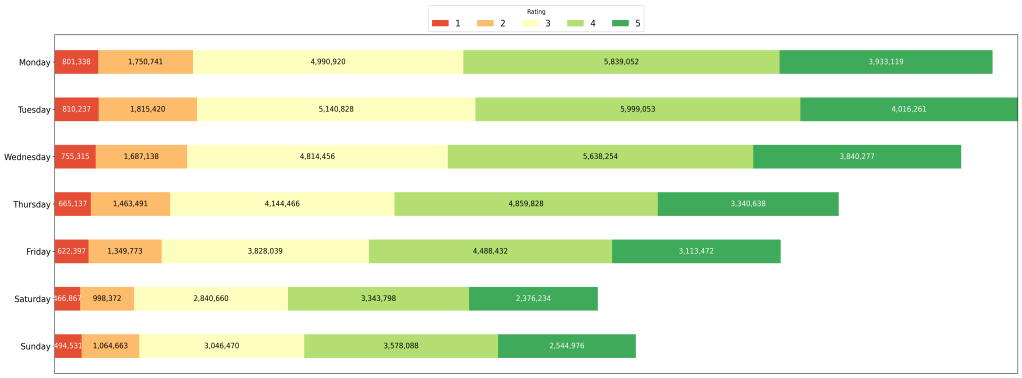
(b) Daily Proportion of Rating (Cumulative)



(c) Mean of Rating (Cumulative)



(d) Standard Deviation of Rating (Cumulative)

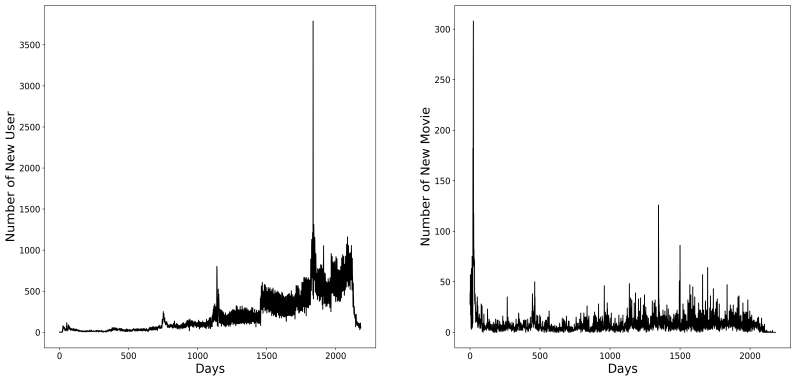


(e) Number of Daily Rating

Source: Author's estimation (2020)

As shown in Figure 1b, the majority of users were initially gave movies rating 3, until rating 4 dominating the proportion of daily rating in the dataset around day 1,000. This temporal changes also affect the cumulative mean of the rating as shown in Figure 1c. Although it shows high fluctuation before day 500, the cumulative mean of rating is

Figure 2: Non-cumulative New Users and Movies



(a) Daily New User (b) Daily New Movie

Source: Author's estimation (2020)

relatively stable afterwards, and increase after day 1,500 due to increase in proportion of rating 5 in the dataset. Moreover, as shown in Figure 1d, the standard deviation for rating also fluctuates before day 500, and shows decreasing trend afterwards. Given the increase in cumulative mean and this decreasing trend in standard deviation of rating data, we could observe users' tendency to give higher rating as time goes. It might also be attributed to better RS's performance. However, since the dataset does not provide what recommendations users were given, we cannot validate this indication. Subsequently, as shown in Figure 1e, majority of rating were given on Tuesday, followed by Monday and Wednesday. Although we cannot find official explanation about this phenomena in the dataset description, we believe it might be attributed to Netflix's DVD rental business process, where users would rent a DVD to be watched over the weekend and provide feedback later on.

Moreover, we also investigate the daily non cumulative of new users and movies in Figure 2. Since we do not have the data on which the user and the movie were added to the system, we consider their first rating date as their onboarding date. As suggested by Lathia (2010), measuring user and movie growths this way is reasonable given CF algorithms can only be queried for recommendation for contents that have been rated and users that have rated at least one content. As we could observe in Figure 2a, more users are added to the dataset on a later date. On the other hand, as shown in Figure 2b, movie growth is relatively constant over time. Subsequently, the cumulative growth of both user and movie provide different perspectives as shown in Figure 3. Due to the more users were added later in the dataset, the user data, as shown in Figure 3a, grows almost exponentially. Furthermore, the movie data in the dataset grows in almost linearly as shown in Figure 3b. These figures indicate how the business of Netflix grows over time, where as the company adds new movie, people start to become interested and decided to subscribe to its services. On the other hand, this phenomena might as well present the challenge for CF algorithm. Without any prior rating history, new item might not be recommended to user and new user might receive false recommendation, because CF algorithm cannot infer any similarities or characteristics based on previous rating. To overcome this problem, we prune our test data to include any user and item that have at least 1 historical rating.

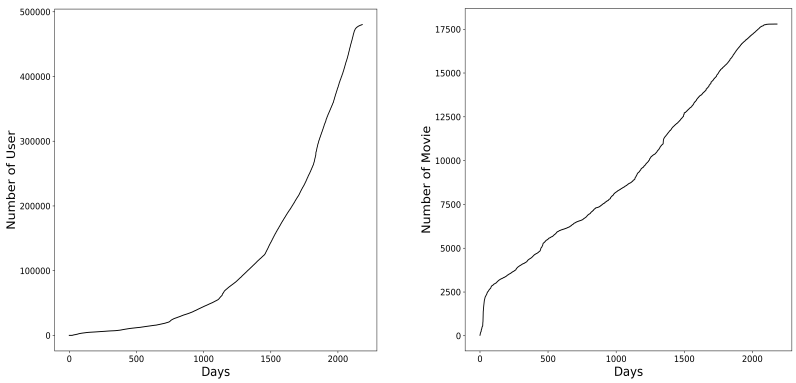
Intuitively, these changes in data distribution would affect the accuracy of RS in production stage, to what extent it drifts is what we investigate in this work. We explain our evaluation procedure in the following Section 4.2.

4.2 Evaluation Procedure

As previously mentioned, we investigate 4 CF algorithms, namely (1) User-based *k*-NN algorithm and (2) Item-based *k*-NN algorithm, (3) SVD algorithm, and (4) SVD++ algorithm. In general, we follow the temporal evaluation technique discussed in Lathia (2010). As previously discussed in Section 1, we first determine the edge time ϵ to day 500 in the dataset $\epsilon = 500$ and system update interval υ to monthly update $\upsilon = 30$-days. Since Lathia (2010) suggests that choosing the edge time ϵ and the update interval υ are arbitrary, we are curious about how these algorithms perform if we keep the edge time and change the update interval to monthly update. The algorithms are then evaluated following the rule that at time τ_t, based on all data available prior to τ_t, is calculated on test set of $\tau_t + \upsilon$ period, repeatedly until the last time point in the dataset. We also prune our test set to include only users and items that have at least 1 historical interaction in the training data. This pruning technique ensures that the algorithms would not have to make predictions for a movie which never rated by any user previously, as well as for a user who never rated any movie previously. Figure 4 graphically illustrate these evaluation protocol.

When an RS is deployed in the production stage, the system would be trained periodically with all available rating history to provide latest recommendation for users. Thus, items to be recommended to users would be affected by users' latest rating activity. Deploying this evaluation technique, we assume that when users rate items, only their latest recommendation would be influencing their decision (Lathia 2010). Moreover, this recommendation process also illustrates the feedback loop in the recommendation system, where users are given recommendation based on their rating input to the system, and the system itself is giving recommendation based on rating data provided by the users. Basilico and Raimond (2017) explain that this feedback loop might lead to degrading quality of the recommendation due to it is challenging to distinguish whether users are responding to the recommendation because they like the items or because it is repeatedly shown to them. However, in this work, we are

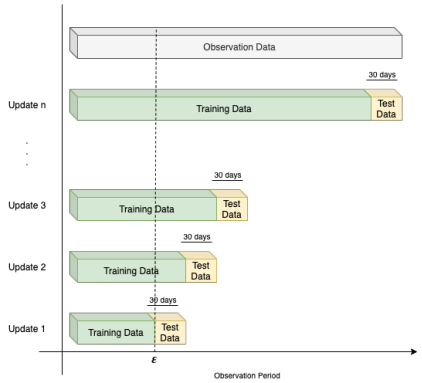
Figure 3: Cumulative User and Movie Growths



(a) User Growth (b) Movie Growth

Source: Author's estimation (2020)

Figure 4: Experiment Design



Source: Author's estimation (2020)

not taking the effect of feedback loop into account when measuring the performance of algorithms.

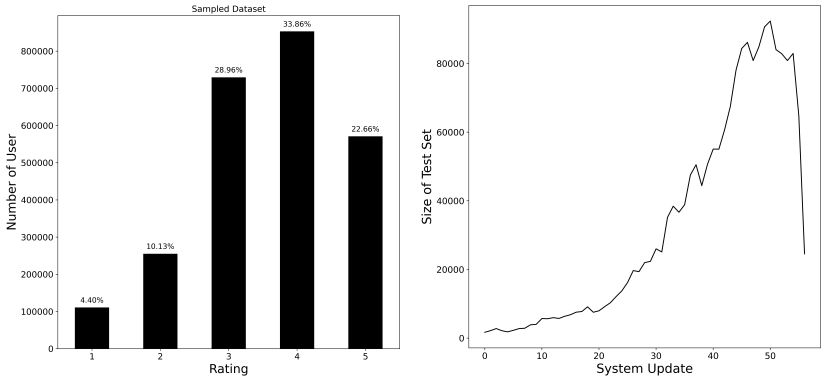
Furthermore, considering the size of Netflix Prize dataset and computing resources available at our disposal, we subsample the dataset using stratified random sampling technique. In doing so, we split the users into 50 categories based on the number of rating they provided (profile size), and randomly choose 2.5% from each category. Thus, our sampled data consists of 2,519,931 rating from 12,006 user on 17,509 movie. As we could observe in Figure 5a, the proportion of each rating relative to the global rating in sampled data is considerably similar to proportion in the full dataset as shown in Figure 1a. Applying previously mentioned evaluation technique in this sampled data allows us to perform 57 system updates. It is also interesting to note that as the size of training set continuously grows as the system updates, the size of test set after pruning also grows as shown in Figure 5b.

Subsequently, we also follow Lathia (2010) to measure temporal performance using 2 methods: *Sequential method*, and *Continuous time-averaged method*. In this sense, sequential method uses RMSE to measure prediction error for each system update independently to previous system updates. On the other hand, continuous time-averaged method uses time-averaged RMSE to measure the prediction error for all prediction made by the algorithms, including from the past system updates: in other words, we measure the dependence of prediction error of algorithms until certain point of a time. Lathia (2010) defines time-averaged RMSE $RMSE_\tau$ is simply the RMSE achieved on the predictions made so far as follows:

$$RMSE_\tau = \sqrt{\frac{\sum_{\hat{r}_{u,i} \in R_\tau}^{N} (\hat{r}_{u,i} - r_{u,i})^2}{|R_\tau|}} \tag{9}$$

In equation 9, R_τ denotes all the prediction made by the algorithms from the first until most recent system update. Incorporating both method will give us insight on the performance at a system update as well as cumulative performance. It is important to note that, we query for prediction only once for each algorithm in each system update. We then evaluate this prediction using both methods. This also implies that there is no observation that being evaluated more than once.

Figure 5: Rating Distribution After Sampling and Size of Test Set



(a) Rating Distribution in Sampled Dataset

(b) Size of Test Set over system update (non-cumulative)

Source: Author's estimation (2020)

As previously mentioned in Section 2, we use $[10, 20, 30, 40, 50]$ as hyper-parameter for both number of k in Neighbourhood methods and number of $factor$ for latent factor models. In addition, we follow Koren and Bell (2015) to set the learning rate and the regularization parameter for SVD algorithm to $[lr = 0.005, reg = 0.02]$ and for SVD++ algorithm to $[lr = 0.007, reg = 0.02]$. For each factor, SVD and SVD++ are trained for 30 epochs, as suggested by Koren and Bell (2015). We are aware that this hyper-parameter setting might not be the optimum setting because we are not performing validation process. However, we found our settings to be justifiable since we focus on observing how the algorithms behave on a temporal situation in this work, and not going to decide which algorithm is the best of all.

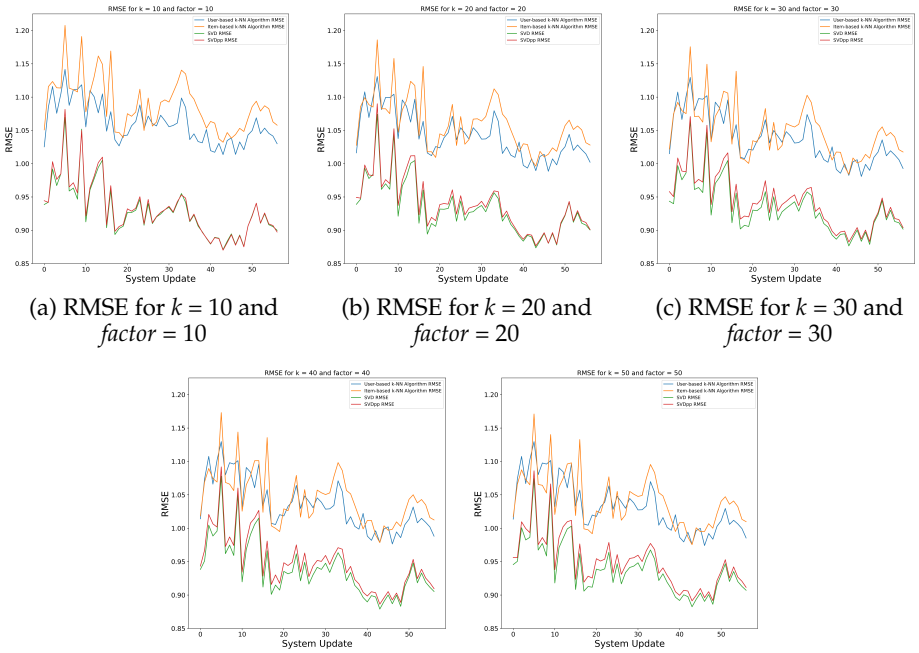
On the technical side, our code for this work is written in Python. We use libraries such as Pandas (Pandas development team 2020) for data management, Numpy (Oliphant 2006) for mathematical operation and computing performance metrics, and Matplotlib (Caswell et al. 2020) for data visualization. We also use Surprise (Hug 2017) library for the algorithms we evaluate. Due to safety and portability reasons, we initially plan to perform our experiment using Google Colaboratory environment as it provides cloud computing services for free. However, due to limitation in parallel processing, we partially execute SVD++ algorithm on Macbook Pro with 2.3 GHz 8-Core Intel Core i9 and 16GB of RAM. To ensure our results to be fully reproducible, we publish our code and all figures we use in this work on GitHub[3]. We present our result in the following Section 5.

5. Result

After we perform temporal performance evaluation using techniques we discussed in the previous section, we found that it requires at least 8 hours to train and evaluate an algorithm from system update 1 to system update 57 on Google Colaboratory environment. To expedite the evaluation process, we also execute our code on our own machine, thus allows us to perform parallel processing. Using this setting, we found that it takes 3

3 Available at https://github.com/miftahulridwan/Temporal-Performance-Evaluation-CF-algorithms.

Figure 6: Sequential RMSE



(a) RMSE for k = 10 and *factor* = 10

(b) RMSE for k = 20 and *factor* = 20

(c) RMSE for k = 30 and *factor* = 30

(d) RMSE for k = 40 and *factor* = 40

(e) RMSE for k = 50 and *factor* = 50

Source: Author's estimation (2020)

days and 6 hours for SVD++ algorithm on system update 57 to finish evaluation process, with other system updates require roughly the same time. We present our results in both sequential (Section 5.1) and time-averaged results (Section 5.2)

5.1 Sequential Result

We plot the sequential results for both neighbourhood methods and latent factor models as shown Figures 6. From these figures, we could observe that for Item-based k-NN algorithms, the higher the number of neighbour, the better the performance in any given system update. On the other hand, while the performances for $k \neq 10$ in User-based k-NN algorithm overlap with each others, the result for $k = 10$ consistently shows highest RMSE, meaning lowest performance, in any given system update. Moreover, the result for SVD algorithm shows a pattern that the higher the number of factor negatively impacted the performance, whereas $factor = 10$ outperformed other number of factors in 47 out of 57 system updates. The result for SVD++ algorithm also presents similar pattern where the lower the number of factor, the better the performance. Particularly, the result for $factor = 10$ outperformed other number of factors in 50 out of 57 system updates. The only system update where higher number of factor perform worse than the lower number of factor is in the first update where the result for $factor = 40$ (0.9434) outperformed the performance of other number of factors.

Furthermore, we identify 14 system updates where the performance of all algorithms simultaneously drop in any given number of neighbour and factor. They are update 2, 11, 16, 23, 25, 28, 32, 33, 44, 47, 49, 50, 51, and 53. The most notable drop is on update 16, where the performance drop ranging from 2.31% (User-based k-NN

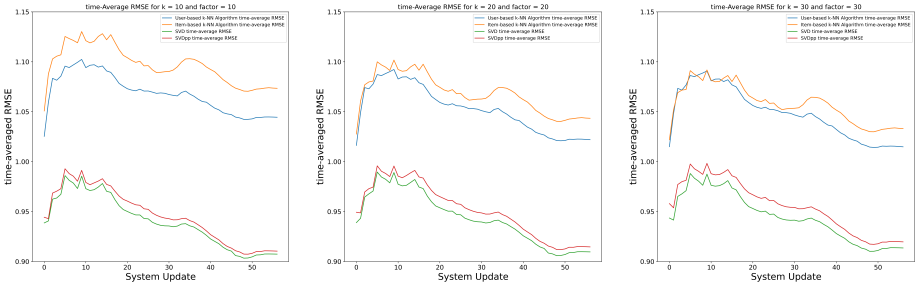
algorithm for $k = 50$) to 11.15% (Item-based *k*-NN algorithm for $k = 50$). This drop in performance signifies the changes in the rating distribution, considering the changes in majority rating proportion in days 1000 in the dataset from rating 3 to rating 4, around the same time as test set for system update 16. Moreover, as we could also observe in Figure 6, SVD algorithm consistently perform the best across all number of neighbour and factor we try. However, the average distance between the neighbourhood methods and latent factor models is getting closer as the number of neighbour and factor increases.

We could also observe the closest distance between the two methods happen between User-based *k*-NN algorithm for $k = 50$ and SVD++ algorithm for $factor = 50$ in update 9 which is 0.0350 difference in RMSE. On the other hand, the furthest distance is happen between Item-based *k*-NN algorithm for $k = 10$ and SVD algorithm for $factor = 10$ in Update 16 which is 0.2088 difference in RMSE. These details align with our earlier observation that for the neighbourhood methods, the higher the number of neighbour the better the performance, and *vice versa* for the latent factor models, the higher the number of factor, the lower the performance.

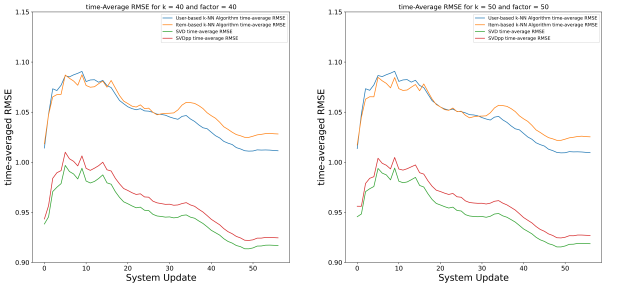
Moreover, we report the sequential result in Table 1 in Appendix A. From this table, we could observe that the performance CF algorithms lies on a range of values, as noted by Lathia (2010). Moreover, the range of performance for an algorithm depends on the number of neighbour and factor being selected. In general, for number of neighbours $[10, 20, 30, 40, 50]$, the prediction error of neighbourhood methods lies on the range of $[0.9741, 1.2077]$, with the range of prediction error for User-based *k*-NN algorithm lies on the range of $[0.9741, 1.1416]$ and the prediction error for Item-based *k*-NN algorithm lies on the range of $[0.9754, 1.2077]$. Whilst the worst performance of this methods happen in Update 5 for all number of neighbours, the best performance for Item-based *k*-NN algorithm in all number of neighbour and User-based *k*-NN algorithm for $k = 10$ happen on the update 43. On the other hand, the lowest prediction error of other number of neighbour in User-based *k*-NN algorithm happen on update 46. Furthermore, for number of factor $[10, 20, 30, 40, 50]$, the prediction error of latent factor models lies on the range of $[0.8701, 1.0918]$, with the prediction error for SVD algorithm lies on the range of $[0.8709, 1.0779]$ and the prediction error for SVD++ algorithm lies on the range of $[0.8701, 1.0918]$. Whilst the lowest RMSE for both algorithms happen at update 43, their worst performance happen at update 5.

It is important to note that we do not find any evidence that the neighbourhood methods outperformed the latent factor models, neither at any given system update, nor at any number of neighbour and number of factor. It is different from what Lathia (2010) highlights that the performance of Item-based *k*-NN algorithm overlap with SVD algorithm. We understand this due to the differences in update interval in this research, sampling methods, and number of factor selected for the SVD algorithm. Therefore, the result of this research confirms the claim made by Koren and Bell (2015) that latent factor models perform better than the neighbourhood methods. However, we do found that SVD++ algorithm does not always perform better than SVD algorithm as suggested by Koren and Bell (2015), even using the same number of factor for both algorithms. In fact, we found that the temporal performance of SVD algorithm is better than SVD++ algorithm for all $factor \neq 10$. This contradiction is highlighted by Lathia (2010) that it is challenging to claim one algorithm outperformed the others when only a static case is investigated. Therefore, how an algorithm perform over time should always be investigated before claiming its superiority over other algorithms. From here, we proceed to discuss the time-averaged result in the following Section 5.2.

Figure 7: Sequential RMSE



(a) Time-averaged RMSE for $k = 10$ and *factor* = 10 (b) Time-averaged RMSE for $k = 20$ and *factor* = 20 (c) Time-averaged RMSE for $k = 30$ and *factor* = 30



(d) Time-averaged RMSE for $k = 40$ and *factor* = 40 (e) Time-averaged RMSE for $k = 50$ and *factor* = 50

Source: Author's estimation (2020)

5.2 Time-averaged Result

We plot the time-averaged results for both neighbourhood methods and latent factor models as shown Figures 7. The movement of the time-average RMSE means the changes on *quality* of the training set in which the algorithms are trained upon. Similar to the sequential result, time-averaged result also present the same pattern that, for neighbourhood methods, the higher the number of neighbour, the better the performance, and *vice versa*, for the latent factor models, the higher the number of factor, the worse the performance. Moreover, we also observe 9 updates $[2, 9, 12, 14, 33, 34, 50, 51, 53]$ in which the time-averaged RMSE of all algorithms simultaneously falls at any given number of neighbour and number of factor. The most notable fall happens on update 9, where the performance drop ranging from 1.38% (Item-based k-NN algorithm for $k = 10$) to 2.85% (SVD++ algorithm for $factor = 40$). In general, all algorithms present converge trend towards better performance as the training set expand.

Similar to sequential result — we found no evidence that the performance of neighbourhood methods overlap with the performance of latent factor models, at any given system update and number of neighbour and number of factor — in time-averaged result. One interesting observation we could make is that the difference between the two methods, on all number of neighbour and number of factor, reach its lowest point on the first update. The closest difference of performance happen also between User-based k-NN algorithm for $k = 30$ and SVD++ algorithm for $factor = 30$ with 0.0570 difference in time-averaged RMSE on the first update. The furthest distance, on the other hand, also happen between Item-based k-NN algorithm for $k = 10$ and SVD algorithm for $factor = 10$ with 0.1698 difference in time-averaged RMSE on update 41.

We report the time-averaged result in Table 2 in Appendix A. From this table, we could observe that the best and the worst performance for each algorithm and each hyper-parameter happen at different system update. The best performance for User-based k-NN algorithm for $k = [10, 20]$ happens in the first update, and for the rest of the tested k, it happens at update 49. On the other hand, the worst performance for SVD algorithm for $k \neq 50$ happens on update 5, and for $k = 50$, it happens later on update 9. This indicate how the hyper-parameter setting inside the algorithm cope with difference in quality of training data, an interesting topic we plan to address in the future research.

6. Discussion

In this work, we focus on investigating the performance of CF algorithms when evaluated using temporal settings. We aim provide better understanding about the performance drift when CF algorithms are deployed in production stage, thus help the businesses to design and/or improve their existing RS to provide better recommendation to the customer. Indeed, better recommendation will reduce our exposure to information overload and, on a greater extent, improve our overall quality of life. In doing so, we evaluate 4 CF algorithms that belongs to 2 main disciplines of CF: the neighbourhood methods (User-based and Item-based k-NN algorithms) and the latent factor models (SVD and SVD++ algorithms).

Considering the sequential result and the time-averaged result, we found that the prediction error for User-based *k*-NN algorithm lies $[0.9741, 1.1416]$ and the prediction error for Item-based *k*-NN algorithm lies on the range of $[0.9754, 1.2077]$. Moreover, we also found that the prediction error for SVD algorithm lies on the range of $[0.8709, 1.0779]$ and the prediction error for SVD++ algorithm lies on the range of $[0.8701, 1.0918]$. Whilst Lathia (2010) suggests that performance of the neighbourhood methods depend on proper value of k being selected, we found that it tends to get better as higher number of neighbour increase. Moreover, we found that the performance of latent factor models decrease as number of factor increases. However, we found no evidence that these performance overlap with the latent factor models, at any given system update, using any given hyper-parameter setting we tested. Although it is contrary to what found in Lathia (2010), the authors uses $[20, 35, 50]$ as number of neighbour and 65 as number of factor for SVD algorithm. Thus, as the performance pattern suggests, it makes sense if at one point, the performance of SVD and User-based *k*-NN will overlap with each others.

Furthermore, we observe performance overlap between latent factor models. In this sense, SVD++ algorithm does not always perform better than SVD algorithm as suggested by Koren and Bell (2015), even using the same number of factor for both algorithms. In fact, we found that the temporal performance of SVD algorithm is better than SVD++ algorithm for all $factor \neq 10$. We understand this phenomena as the effect of our choice on learning rate, regulatization parameter, and number of epochs. SVD and SVD++ algorithms infer movie and user characteristics via stochastic gradient descend. They first initialize random value as many as the number of factor being included, and adjusted the value based on learning rate parameter and regularization, for as often as number of epochs. Although we follow the learning rate, regularization parameter and number of epochs as suggested in Koren and Bell (2015), different structure of data require different optimum hyper-parameter setting. The process of tuning hyper-parameter setting usually happen in the validation process. However, since we aim to study how these algorithms' performances change over time, and not deciding which algorithm perform the best, we do not perform validation process. We consider our

hyper-parameter setting to be justifiable, given we follow the settings of the authors who claim performance superiority.

Insights from this work are not observed when performing static evaluation technique. Indeed, the work of Lathia (2010); Soto (2011) and Beel (2017) suggest that performance evaluation over time should always be calculated and presented to better understand the behaviour of CF algorithms. Thus, we echo the message from these authors to always evaluate algorithm's performance in temporal settings before claiming one algorithm is better than the others. Because, the performance of an algorithm depends on which snapshot of data it is trained and evaluated upon and the hyper-parameter settings. Moreover, this work is contribute to the RS literature by performing 3 widely discussed algorithms, namely (1) User-based k-NN, (2) Item-based k-NN and (3) SVD algorithms. We incorporate SVD++ algorithm since it is the least discussed algorithm in time-aware evaluation setting. However, we are aware that all algorithms we evaluate in this work are not incorporating time-information explicitly in the model. We plan to incorporate these algorithms in the future work.

7. Conclusion

This work evaluate temporal performance of CF algorithms by incorporating a sequence of training and evaluation process on expanding and dynamic dataset. The performance of all algorithms we evaluate in this research lies on a range of values. For number of neighbours $[10, 20, 30, 40, 50]$, the performance for User-based k-NN algorithm lies on the range of $[0.9741, 1.1416]$ and the performance for Item-based k-NN algorithm lies on the range of $[0.9754, 1.2077]$. On the other hand, for number of factor $[10, 20, 30, 40, 50]$, the performance for SVD algorithm lies on the range of $[0.8709, 1.0779]$ and the performance for SVD++ algorithm lies on the range of $[0.8701, 1.0918]$.

This work indicates the urge to always perform temporal performance evaluation before claiming one algorithm is better than the others. Moreover, future research could work on investigating other performance superiority over other algorithms on temporal settings. Particularly, how continuous time-aware algorithms (e.g. time-weight decay algorithm in Ding and Li (2005)) perform over time. Indeed, understanding how RS algorithm perform over time help the businesses to design and/or improve their existing RS to provide better recommendation to the customer, thus reduce our exposure to information overload and, on a greater extent, improve our overall quality of life.

References

Baltrunas, Linas and Xavier Amatriain. 2009. Towards time-dependant recommendation based on implicit feedback. In *Workshop on context-aware recommender systems (CARS'09)*, pages 25–30, Citeseer.

Basilico, Justin and Yves Raimond. 2017. Déja vu: The importance of time and causality in recommender systems. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 342–342.

Beel, Joeran. 2017. It's time to consider "time" when evaluating recommender-system algorithms [proposal]. *arXiv preprint arXiv:1708.08447*.

Campos, Pedro G, Fernando Díez, and Iván Cantador. 2014. Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction*, 24(1-2):67–119.

Caswell, Thomas A, Michael Droettboom, Antony Lee, John Hunter, Eric Firing, David Stansby, Jody Klymak, Tim Hoffmann, Elliott Sales de Andrade, Nelle Varoquaux, Jens Hedegaard Nielsen, Benjamin Root, Phil Elson, Ryan May, Darren Dale, Jae-Joon Lee, Jouni K. Seppänen, Damon McDougall, Andrew Straw, Paul Hobson, Christoph Gohlke, Tony S Yu, Eric Ma, Adrien F. Vincent, Steven Silvester, Charlie Moad, Nikita Kniazev, Paul Ivanov, Elan Ernest, and Jan Katins. 2020. matplotlib/matplotlib: Rel: v3.2.1.

Ding, Yi and Xue Li. 2005. Time weight collaborative filtering. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 485–492.

Eppler, Martin J. and Jeanne Mengis. 2004. The concept of information overload: A review of literature from organization science, accounting, marketing, mis, and related disciplines. *The Information Society*, 20(5):325–344.

Gunawardana, Asela and Guy Shani. 2015. Evaluating recommender systems. In *Recommender systems handbook*. Springer, pages 265–308.

Hug, Nicolas. 2017. Surprise, a Python library for recommender systems. http://surpriselib.com.

Jeunen, Olivier. 2019. Revisiting offline evaluation for implicit-feedback recommender systems. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 596–600.

Jeunen, Olivier, Koen Verstrepen, and Bart Goethals. 2018. Fair offline evaluation methodologies for implicit-feedback recommender systems with mnar data. In *Proceedings of the REVEAL 18 Workshop on Offline Evaluation for Recommender Systems (RecSys '18)*.

Karatzoglou, Alexandros. 2011. Collaborative temporal order modeling. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 313–316.

Koren, Yehuda and Robert Bell. 2015. Advances in collaborative filtering. In *Recommender systems handbook*. Springer, pages 77–118.

Lathia, Neal Kiritkumar. 2010. *Evaluating collaborative filtering over time*. Ph.D. thesis, UCL (University College London).

Oku, Kenta, Shinsuke Nakajima, Jun Miyazaki, and Shunsuke Uemura. 2006. Context-aware svm for context-dependent information recommendation. In *7th International Conference on Mobile Data Management (MDM'06)*, pages 109–109, IEEE.

Oliphant, Travis. 2006. NumPy: A guide to NumPy. USA: Trelgol Publishing, http://www.numpy.org/. [Online; accessed May 29, 2020].

Potter, Gavin. 2008. Putting the collaborator back into collaborative filtering. In *Proceedings of the 2nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*, pages 1–4.

Soto, Pedro G Campos. 2011. Temporal models in recommender systems: an exploratory study on different evaluation dimensions. Master's thesis, Universidad Autónoma de Madrid Escuela Politécnica Superior.

Pandas development team, The. 2020. pandas-dev/pandas: Pandas.

Thakkar, Priyank, Krunal Varma, Vijay Ukani, Sapan Mankad, and Sudeep Tanwar. 2019. Combining user-based and item-based collaborative filtering using machine learning. In *Information and Communication Technology for Intelligent Systems*. Springer, pages 173–180.

Thorat, Poonam B, RM Goudar, and Sunita Barve. 2015. Survey on collaborative filtering, content-based filtering and hybrid recommendation system. *International Journal of Computer Applications*, 110(4):31–36.

Appendix A: Result Details

Table 1: RMSE of all algorithms for *k* and *factor* = [10, 20, 30, 40, 50]. The lowest and the highest RMSE are in Bold

		Best Performance		Worst Performance		
Algorithm	*k & factor*	RMSE	Update	RMSE	Update	Average RMSE
User-based *k*-NN Algorithm RMSE	10	1.0137	43	**1.1416**	5	1.0604
	20	0.9887	46	1.1309	5	1.0423
	30	0.9809	46	1.1295	5	1.0371
	40	0.9765	46	1.1297	5	1.0349
	50	**0.9741**	46	1.1296	5	1.0337
Item-based *k*-NN Algorithm RMSE	10	1.0322	43	**1.2077**	5	1.0892
	20	0.9963	43	1.1859	5	1.0599
	30	0.9842	43	1.1757	5	1.0497
	40	0.9785	43	1.1731	5	1.0448
	50	**0.9754**	43	1.1710	5	1.0418
SVD Algorithm RMSE	10	**0.8709**	43	1.0696	5	0.9298
	20	0.8736	43	1.0741	5	0.9328
	30	0.8765	43	1.0650	5	0.9342
	40	0.8790	43	**1.0779**	5	0.9384
	50	0.8825	43	1.0743	5	0.9392
SVD++ Algorithm RMSE	10	**0.8701**	43	1.0816	5	0.9322
	20	0.8766	43	1.0900	5	0.9387
	30	0.8824	43	1.0708	5	0.9431
	40	0.8864	43	**1.0918**	5	0.9487
	50	0.8913	43	1.0860	5	0.9496

Source: Author's estimation (2020)

Table 2: Time-averaged (TA) RMSE of all algorithms for *k* and *factor* = [10, 20, 30, 40, 50]. The lowest and the highest TA RMSE are in Bold

		Best Performance		Worst Performance		Average
Algorithm	*k & factor*	TA RMSE	Update	TA RMSE	Update	TA RMSE
User-based *k*-NN Algorithm TA RMSE	10	1.0254	0	**1.1024**	9	1.0679
	20	1.0163	0	1.0923	9	1.0523
	30	1.0143	49	1.0908	9	1.0483
	40	1.0111	49	1.0907	9	1.0467
	50	**1.0094**	49	1.0908	9	1.0460
Item-based *k*-NN Algorithm TA RMSE	10	1.0508	0	**1.1303**	9	1.0968
	20	1.0277	0	1.1018	9	1.0680
	30	1.0219	0	1.0919	9	1.0580
	40	1.0184	0	1.0873	9	1.0531
	50	**1.0172**	0	1.0847	5	1.0502
SVD Algorithm TA RMSE	10	**0.9034**	48	0.9861	5	0.9401
	20	0.9059	48	0.9896	5	0.9435
	30	0.9103	48	0.9882	5	0.9447
	40	0.9136	49	**0.9969**	5	0.9492
	50	0.9155	49	0.9942	9	0.9496
SVD++ Algorithm TA RMSE	10	**0.9074**	49	0.9929	5	0.9461
	20	0.9120	49	0.9958	5	0.9512
	30	0.9173	49	0.9983	9	0.9554
	40	0.9220	49	**1.0100**	5	0.9607
	50	0.9245	49	1.0048	9	0.9610

Source: Author's estimation (2020)