Time-aware Collaborative Topic Regression: Towards Higher Relevance in Textual Item Recommendation

Anas Alzogbi

Department of Computer Science, University of Freiburg
79110 Freiburg, Germany
alzoghba@informatik.uni-freiburg.de

Abstract. Time is an important aspect in Recommender Systems. Its impact is observed in several aspects ranging from the change in user interest to the dynamics of adding new users and items into the system. In this work, we present a time-aware recommender system that accounts for the concept-drift in user interest. By computing user-specific concept-drift score, our model controls which ratings should have more influence in the process of learning the recommender model. We consider the use-case of scientific papers recommendation and conduct experiments on a real-world dataset from citeulike. The results clearly show the superiority of the proposed model over the state-of-the-art methods. They additionally show that conducting time-aware evaluations is essential to achieve realistic evaluation for the recommender system.

Keywords: Time-aware RS; Hybrid Recommendation; Latent Dirichlet Allocation (LDA); Matrix Factorization; Scientific paper recommendation.

1 Introduction

Collaborative filtering (CF) in general and matrix factorization in particular has gained a lot of attention in the last decade as a recommendation technique. Since matrix factorization (MF) showed promising results in generating recommendations [10], more and more works engaged this method for CF Recommenders. A successful approach that builds on matrix factorization and recently gained considerable interest is Collaborative Topic Regression (CTR) for recommending scientific articles [19]. CTR leverages not only collaborative ratings but also articles' textual content in order to learn the latent models for users and items. Several works pushed CTR further in different directions. For example, adapting CTR to consider item tags [20], employing autoencoders for a better latent topic modeling [21], or considering the word order in the textual content [2]. Although these works demonstrate appealing results, conducted evaluations ignore an important aspect, the temporal nature of recommender systems. Offline evaluations that don't respect the chronological order of users ratings in the process of train/test data splitting, allow the model to learn from future data, i.e., when

the split procedure doesn't guarantee that all training data points are prior in time compared to test data points. We call such evaluations "time-ignorant" and those which obey the temporal order "time-aware" evaluations. Previous works [4,14] showed that conducting time-ignorant evaluations promise unrealistic performance, whereas time-aware evaluations can better simulate real-world scenarios and provide therefore more realistic results. The difference in results between time-aware and time-ignorant evaluations can be explained by the "concept-drift" in user interest, i.e., the change of user interest over time.
In this paper, we show that the performance of CTR drops significantly when evaluated under a time-aware evaluation framework over a real-world dataset. This motivates on the one hand, applying time-aware evaluations to assess the quality of a recommender system and on the other hand, extending CTR to consider temporal aspects, which is our main contribution in this work.
Concept-drift in user interest over time is a widely known aspect when building real-world recommender systems. It can be observed in various applications, for example: news, books and scientific papers recommendations. We distinguish between two models for temporal influence over the behavior of users: (a) time as context [18] where user habits repeat regularly at certain intervals. Here, the time value (weekend, evening, summer, etc.) when computing predictions plays an important role in deciding the user interest; and (b) time as an aging factor, where time diminishes old user interactions (ratings). As time elapses, old user's interactions become less representative for the actual user interest. In contrast to the "time-as-context" model, here, the *age* of the user interaction decides its importance in defining actual user interest. Concept-drift is related to the latter model and in this work we look at the time from this perspective, time as an aging factor which is the motor for the drift in user interest.
The role of concept-drift in recommender systems has been addressed by a wide range of previous works [18]. A common strategy is to apply forgetting mechanism, in which old ratings are down weighted so that their contribution in computing the actual user model is penalized. This is achieved by using a time-decay function to compute a weight for each rating. The older the rating is, the lower the corresponding weight gets. However, the steepness of the decay function is regulated by a damping (forgetting) factor and the specific value of this factor is usually set emperically as in [17,14,1,5,11].
In this work, we bring the time aspect to CTR. We present Time-aware Collaborative Topic Regression (T-CTR), a recommendation method that applies a forgetting strategy to account for the concept-drift in user interest over time. In contrast to existing works, in T-CTR, we emphasize the fact that users have different dynamics when it comes to the interest drift, some users tend to change their interest faster than others. Therefore, we suggest to compute a personalized concept-drift score for each user, a score that quantifies the user tendency to change his/her interest as time goes on. Then, we utilize user concept-drift score as a forgetting factor to compute a weighting value for each observed rating. The main contributions of this paper can be summarized in:

- A time-aware hybrid recommender system for textual items (items associated with text content) that dynamically accounts for the concept-drift in user interest by leveraging the textual content of rated items.
- An experimental study on a real-world dataset that explores the differences between time-aware and time-ignorant evaluation methods when evaluating recommender systems.
- A real-world dataset that enables conducting time-aware offline evaluations.

The remainder of this paper is organized as follows: in Section 2, we review the most relevant existing works; Afterwards, in Section 3 we introduce our notation and the important preliminaries; in Section 4, we present our method; then, we explain the conducted experiments and analyze the findings in Section 5; finally, we conclude in section 6.

2 Related Work

Several works addressed the role of time and concept drift in recommender systems [18]. Koren introduced in [9] a matrix factorization method that learns time-based biases along learning users and items latent factors in a method called timeSVD++. The time period of the ratings is divided into bins (time intervals) and a bias is learned for each bin. This method can compute predictions for time intervals only if they appear in the training phase, it is therefore not applicable in time-aware evaluation setup. A recent approach fits a time series model that learns from historical ratings how users latent models evolve over time as in [12,13,6]. This approach involves refitting the latent models at each time interval and afterwards fitting the auto-regressive model that finds the linear correlation between actual user latent model and the previous ones. This process adds an extra complexity on the recommendation algorithm. Additionally, as we will show in Subsection 5.4, these methods don't produce good results when the underlying data has few ratings within small intervals.
Another strategy for considering temporal influence which is similar to ours, is to apply a forgetting mechanism in which old ratings are either discarded or down weighted based on a *forgetting factor* [5,1,17,11]. In these works, the forgetting factor is set empirically, whereas in our work, we compute an individual value for each user dynamically (cf. Subsection 4.1). Time aspect in matrix factorization was also addressed in [14], the work suggested a stream-based algorithm for updating users and items preference models in an online fashion. As new ratings arrive, old ratings are considered obsolete and this triggers either refitting the learned models or penalizing old models. The authors suggested also several strategies for dealing with old ratings. A key difference in our work is that we leverage the items textual content for estimating user-specific concept-drift scores.

3 Problem Statement and Preliminaries

Before explaining the details of our method, we introduce some notation and give a brief explanation about important background information relevant to our method: matrix factorization and collaborative topic regression.

3.1 Notation and Problem Statement

Let $\mathcal{U} = \{u_1, \ldots, u_n\}$ be the set of users and $\mathcal{I} = \{i_1, \ldots, i_m\}$ the set of items. We assume each item has textual content and is associated with a bag of words representation over the set of domain-related vocabulary. Additionally, each user has a set of *relevant* items, recorded in the rating matrix $R \in \mathbb{R}^{n \times m}$. An entry R_{ui} has a value of 1 if the user u is interested in item i, otherwise $R_{ui} = 0$. We assume the one-class scenario where only relevant items are known. Therefore, zero values in R don't necessarily represent negative ratings but also unknown ratings. Each rating R_{ui} is associated with a time stamp $t_{R_{ui}}$ that records the time when user u rated item i. Given $\mathcal{U}$, $\mathcal{I}$, and R, the goal is to predict for every user $u \in \mathcal{U}$ at a given time T, the set of top M relevant items from $\mathcal{I}$.

3.2 Matrix Factorization for Collaborative Filtering

Matrix factorization (MF) is one of the most successful recommendation methods for model-based collaborative filtering [10,15]. The main idea is to factorize the incomplete rating matrix R into two matrices with a joint latent low-dimensional space of dimension k: the users latent matrix $U \in \mathbb{R}^{n \times k}$ and the items latent matrix $V \in \mathbb{R}^{m \times k}$, where each user u and each item i are represented as latent vectors $U_u \in \mathbb{R}^m$, $V_i \in \mathbb{R}^n$ respectively.

Zero values in the rating matrix don't necessarily denote negative ratings, but also unknown or missing ratings. Therefore, they should have less contribution in the learning process in comparison to known ratings. This is the well-known *one-class problem*, where only positive ratings are available. To solve this problem, confidence weights are introduced [16,7] where zero values are weighted by a small value b and non-zero values are weighted by a larger value a such that $a > b > 0$. Given R, a matrix factorization algorithm finds U and V that minimize the following objective function with confidence weights, the regularized squared reconstruction error:

$$\underset{U,V}{\operatorname{argmin}} \sum_{u \in \mathcal{U}, i \in \mathcal{I}} C_{ui}(R_{ui} - U_u^T V_i)^2 + \lambda_u \sum_{u \in \mathcal{U}} ||U_u||^2 + \lambda_v \sum_{i \in \mathcal{I}} ||V_i||^2 \tag{1}$$

Where λ_u, λ_v are the regularization parameters and C_{ui} is the confidence weight of the rating R_{ui}:

$$C_{ui} = \begin{cases} a, & \text{if } R_{ui} \neq 0 \\ b, & \text{otherwise} \end{cases} \tag{2}$$

After finding U and V, we can estimate the affinity of user u towards item i by the dot product between their latent factors:

$$\hat{R}_{ui} = U_u^T V_i \tag{3}$$

3.3 Collaborative Topic Regression (CTR)

CTR [19] is a hybrid recommendation approach that builds on MF and extends it to benefit from items' textual content. It adopts matrix factorization for one-class problem. Additionally, it assumes that items' latent vectors are generated

from a topic model, specifically, Latent Dirichlet Allocation (LDA) [3]. LDA is a topic modeling algorithm that finds a set of topics for a set of documents. Let $\theta \in \mathbb{R}^{m \times k}$ be the matrix of k latent topics extracted from a set of m items by LDA, where $\theta_i \in \mathbb{R}^k$ is the topic vector of item i. CTR consists in factorizing the rating matrix R into users latent matrix U and items latent matrix V, such that V is basically the latent topics vectors extracted by LDA with an added offset: $V_i = \theta_i + \epsilon_i$. The offset ϵ_i represents how much of the prediction relies on i's content and how much it relies on other users ratings on item i. To solve the matrix factorization, CTR applies a probabilistic model as in [15] that aims at maximizing the log likelihood of the model variables[1] U and V:

$$\mathcal{L} = -\sum_{u \in \mathcal{U}, i \in \mathcal{I}} \frac{C_{ui}}{2}(R_{ui} - U_u^T V_i)^2 - \frac{\lambda_u}{2}\sum_{u \in \mathcal{U}} ||U_u||^2 - \frac{\lambda_v}{2}\sum_{i \in \mathcal{I}} ||(V_i - \theta_i)||^2 \quad (4)$$

The algorithm to maximize $\mathcal{L}$ alternates the parameter optimization between U and V until convergence. Each time, one parameter is fixed to its current estimate and the other parameter is optimized by differentiation, which leads the following analytic solutions:

$$U_u \leftarrow (V^T C_u V + \lambda_u \boldsymbol{I})^{-1} V^T C_u R_u \quad (5)$$

$$V_i \leftarrow (U^T C_i U + \lambda_v \boldsymbol{I})^{-1} (U^T C_i R_i + \lambda_v \theta_i) \quad (6)$$

Where $\boldsymbol{I}$ is $k \times k$ identity matrix, $C_u \in \mathbb{R}^{m \times m}$ and $C_i \in \mathbb{R}^{n \times n}$ are diagonal matrices, with $C_{u1}, \cdots, C_{um}$ and $C_{1i}, \cdots, C_{ni}$ at their diagonals respectively and $R_u \in \mathbb{R}^m$ is the vector that contains u's preferences. Similarly, $R_i \in \mathbb{R}^n$ is the vector that contains preferences for item i.
After finding U and V, we approximate the missing ratings using Equation 3.

4 Time-aware Collaborative Topic Regression (T-CTR)

For considering the temporal aspect, we propose T-CTR. Our approach is a hybrid recommender system that is capable of accounting for concept-drift in user interest. It learns users and items latent models seamlessly from items' textual content and users ratings. Additionally, we impose the time influence in the model by extending the role of confidence weights. As we have seen in the previous section, confidence weights are employed to give known ratings more importance than unknown ratings. In T-CTR, we give confidence weights an additional task, which is expressing different importance levels for different *known* ratings. As mentioned earlier, the older a rating gets, the less important it becomes in representing the actual user interest. Therefore, we make old ratings weigh less than recent ones. Above that, the aging process of ratings is user-specific i.e different users bare different concept-drift mechanisms. For example,

[1] In CTR, the matrix of items latent topics θ is considered as a variable as well, but we removed it from the list of variables for simplicity because experiments in [19] showed that fixing θ as the result of LDA gives comparable performance.

given two users A and B where A tends to change the topics of interest more rapidly than B. Assume that both users have 6-months-old ratings: $R_{A,i}$, $R_{B,j}$ respectively. Although these ratings have the same age, knowing that A tends to change the topics of interest more rapidly than B gives $R_{B,j}$ more importance in representing the actual interest model of B than $R_{A,i}$ in representing the actual interest model of A. In order to account for this difference in users' behavior, we calculate a per-user concept-drift score. This score quantifies the user tendency to change his/her interests as time goes on. User's concept-drift score is then involved in computing the ratings confidence weights, which allows getting different confidence weights for ratings from different users even when they have the same age. In the following, we explain in details the users concept-drift scores, ratings weights and the model learning algorithm.

4.1 Concept-drift Score

The inter-similarity between items that successively appear in the user's list of relevant items gives us an important evidence whether the user has a consistent taste or tends to show a drift in the interest. The lower this similarity is, the higher the likelyhood that the user experiences a concept-drift in her interest. In order to calculate similarities between items, we choose to represent items using their latent topics. Therefore, given the items textual content, we extract the set of k latent topics for each item using LDA. Let $\theta \in \mathbb{R}^{m \times k}$ be the items-topics matrix computed by LDA. θ_{il} is the probability of item i having topic l. For each item, we keep only the representative topics, those topics which probability is higher than a certain threshold τ. Γ_i is the set of such topics. In our experiments, we chose $\tau = 0.01$ empirically:

$$\Gamma_i = \{l \mid \theta_{il} \geq \tau\}; i \in \mathcal{I}$$

Given the rating matrix R and Γ, we calculate the concept drift scores as shown in Algorithm 1. For each user u, we first order u's ratings by the rating date. Then, based on item's topics, we calculate the pairwise similarity between each two items i and j that appear successively in u's ratings. In our implementation, we used the Jaccard similarity to calculate the similarity between two sets of topics. The concept-drift score is then calculated as (1 - average pairwise similarity). The result is the set of concept-drift scores for all users $\mathcal{S}$, which will be used in computing confidence weights.

4.2 Ratings Confidence Weights

The confidence weight of a rating quantifies the rating's importance in representing user interest at a given time T and serves to control how much a rating R_{ui} should contribute in the process of learning the latent models of user u and item i. Having the concept-drift scores $\mathcal{S}$ for all users, we apply an exponential decay function (Equation 7) to compute the confidence weights W_{ui} for all u's ratings based on the rating's age: $T - t_{R_{ui}}$. Here, the concept-drift score $\mathcal{S}_u$ controls the steepness of the decay function and it therefore plays the role of

Algorithm 1: ConceptDrift

Input: Items' LDA topics Γ, Rating matrix R
Result: List of users' concept-drift scores $\mathcal{S}$
Initialize $\mathcal{S}$ to an empty list;
for $u \in \mathcal{U}$ **do**
  $P := \{i | R_{u,i} = 1\}$;
  Sort P by the ratings dates;
  initialize $\mathcal{S}_u$ to 0;
  for $i = 1$ *to* $|P| - 1$ **do**
    $\mathcal{S}_u := \mathcal{S}_u +$ Jaccard-similarity$(\Gamma_{P_i}, \Gamma_{P_{i+1}})$;
  end
  $\mathcal{S}_u := \mathcal{S}_u / (|P_u| - 1)$;
  Append $1 - \mathcal{S}_u$ to $\mathcal{S}$;
end

an aging factor, the higher the score is, the steeper the function gets. Figure 1 demonstrates the influence of different values for the concept-drift on the confidence weights. This is a desired behavior to account for the difference in users concept-drift mechanisms. This way, users with higher concept-drift scores, will have steeper curve and as a result, their old ratings get lower weights. The rating's age granularity can be configured based on the underlying application. For example, in the scenario of paper recommendation, we chose the age to be in months.

$$W_{ui} = \frac{2}{1 + e^{\mathcal{S}_u(T - t_{R_{ui}})}} \tag{7}$$

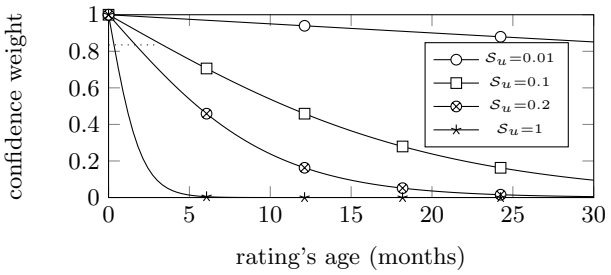

Fig. 1: The impact of the concept-drift score on the decay function.

4.3 Model Learning and Prediction

After computing the concept-drift scores and ratings confidence weights, we can learn the latent topic vectors U and V from R and θ similarly to CTR as explained in Subsection 3.3. But, the confidence scores are not taken from Equation 2, we use our calculated confidence weights instead. Thus, the confidence

matrix C is defined as following:

$$C_{ui} = \begin{cases} \max(W_{ui}, b) & \text{if } R_{ui} = 1 \\ b, & \text{otherwise} \end{cases}$$

Here, b is the confidence score for the unknown ratings, $\{R_{ui} \mid R_{ui} = 0\}$ and is set to a small value as in [16,19].
After finding U and V, we approximate the predicted ratings using Equation 3.

5 Experiments and Discussion

We conducted offline evaluations on a real-world dataset to demonstrate the effectiveness of our model and compare it against other state-of-the-art and basic approaches[2]. In this section, we introduce the used dataset and the experimental setup. Then, we explain the conducted experiments and discuss the findings.

5.1 Dataset

We used a dataset from citeulike[3]. Citeulike allows users to create personalized digital libraries where they can bookmark and tag relevant scientific publications (papers, books, theses,...). Our dataset spans over three years starting from November 2004 to December 2007 and contains information about 210,137 papers and 3,039 users with a total of 284,960 ratings. All users have at least 10 papers in their libraries. Ratings are also associated with timestamps which record the time of adding the paper to the user library. We collected publications meta-data such as title, abstract, publication year and keywords. We defined the dataset vocabulary as a set of 19871 words. It comprises all keywords associated with the papers, in addition to 10000 words extracted from the articles' titles and abstracts. We kept only English words with more than 2 letters and applied stop words removal, stemming and finally removed very in-frequent and very frequent words (those appearing in less than 3 documents or more than 90% of all papers).

5.2 Experimental Setup

In order to apply time-aware evaluations that simulate a real-world scenario, we followed recommendations of Campos et al. in [4]. We chose 5 different dates to be the split points. Each two successive dates are 6 months apart. We simulated a real-life scenario where the recommender rebuilds its model at each split date to generate predictions for the next 6 months. This results in 5 folds, one fold for each split date. All ratings before the split date are considered as the fold's training set, ratings from the next 6 months comprise the fold's test set. The test sets contain ratings from users that appear in the training set. This is because our method doesn't address the problem of having new users (the cold-start

[2] Our implementation is available at: https://github.com/anasalzogbi/T-CTR
[3] http://www.citeulike.org

problem). Note that test sets may contain papers unseen in the corresponding training sets. For each fold, we fit the model on the training set and test it on the test set. Table 1 shows the number of users, papers and ratings in each fold for both training and test sets[4]. The recommender generates for each (user, paper) pair a scalar prediction score that represents the paper's relevance to the user. For each user, the papers are ranked based on the prediction score and top M papers are recommended. We evaluate the presented approach based on the following ranking metrics which are typical for evaluating recommender systems: Mean Reciprocal Rank (MRR), Normalized Discounted Cumulative Gain (nDCG@M) [8] and Recall@M [19]. The average of these metrics over all users for each fold is reported.

Fold	Split date	#Users		#Papers		#Ratings	
		Training	Test	Training	Test	Training	Test
1	Sep 2005	484	310	18,988	10913	25,393	13,968
2	Mar 2006	978	564	46,585	17,026	62,795	21,980
3	Sep 2006	1,518	793	83,217	22,326	112,098	30,559
4	Mar 2007	2,105	1,023	122,596	30,830	169,145	39,652
5	Sep 2007	2,716	970	176,977	22,142	243,787	24,119

Table 1: Dataset statistics for each fold.

5.3 Time-aware vs time-ignorant evaluations

In our initial experiment, we evaluate a state-of-the-art system on our dataset following the time-aware scheme. The goal is to study the applicability and the expected performance of such methods in real-world scenarios and show its deviation from the -usually reported- time-ignorant results. As a representative model, we chose CTR [19] (cf. Subsection 3.3). Figure 2 shows the performance of CTR evaluated on time-ignorant and time-aware schemes. The results of all metrics show clearly that the method performance drops significantly when a time-aware evaluation is imposed. We believe the reason for this behavior is related to the concept-drift in users interests, this can be explained as following. In time-ignorant evaluations, training and test ratings are sampled randomly from the set of all available ratings. This allows the model to possibly sample training ratings from different time-slots and learn accordingly. On the contrary, restricting the training ratings to be sampled exclusively from time slots that are prior in time to test ratings, makes it more challenging for the fitted model to predict future ratings correctly.

5.4 T-CTR against baselines

To analyze the performance of our approach (T-CTR), we compare it with the following methods:

[4] The dataset is available for public use at:
http://dbis.informatik.uni-freiburg.de/forschung/projekte/SciPRec

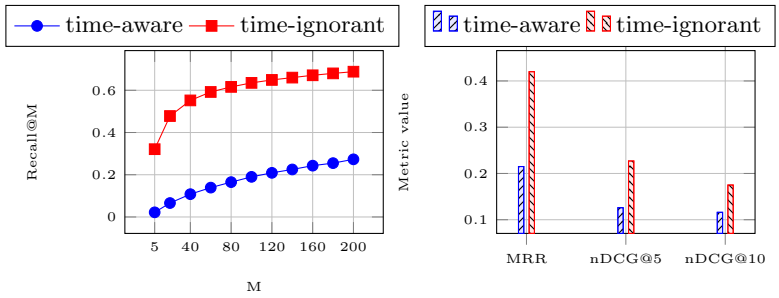

Fig. 2: Performance comparison of time-aware and time-ignorant evaluations for CTR based on Recall@M, MRR and nDCG.

- **CF:** Collaborative Filtering for Implicit Feedback [7] is an effective matrix factorization method for positive-only (one-class) datasets. It factorizes the rating matrix and uses static confidence weights for known and unknown ratings (cf. Subsection 3.2).
- **CTR:** Collaborative Topic Regression [19] performs topic modeling and collaborative filtering simultaneously (cf. Subsection 3.3).
- **CE:** Collaborative Evolution For User Profiling [13]. This work represents the state-of-the-art time-aware MF-based recommender systems. It follows a different strategy than ours, in which the evolution of user latent models over time is learned by fitting an auto-regressive model. Their assumption is, user latent model U_u^t at time t is dependent on the user's previous ϕ latent models $\{U_u^{t-j} \mid 1 \leq j \leq \phi\}$. We chose this method as a representative for such methods [6,12] that learn the evolution of users models instead of applying a forgetting strategy.

Figure 3 shows the evaluation results for all methods on all folds.
CF and CE show the worst performance. CF is based on the rating matrix only and doesn't utilize papers' text. The high sparsity in our dataset and the existence of new papers in the test sets explain the poor performance of those algorithms, which depend on the rating matrix solely.
CE also depends on the rating matrix only. Above that, applying CE in our

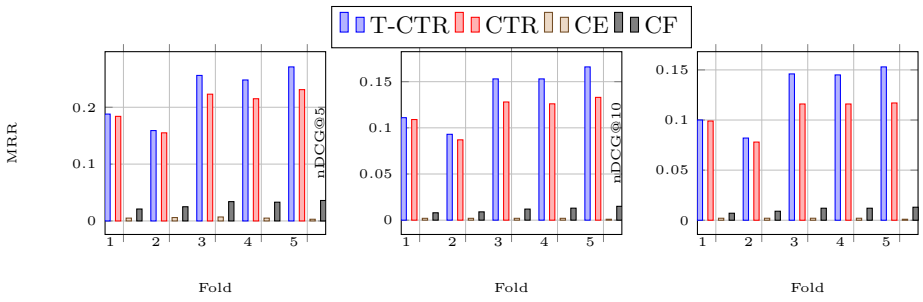

Fig. 3: Performance comparison for T-CTR and the baseline methods. Evaluation metrics are shown for each fold.

scenario shows an additional shortcoming that contributes to its poor performance. Here, we will give more insights about how CE works in order to explain its poor performance on our dataset. In CE, an auto-regressive model is learned from the first T_0 time intervals. It finds ϕ coefficient matrices each is a $k \times k$ matrix. Implementing this method requires several decisions to be made that are subject to the underlying dataset: first, the time interval (day, week, month, etc.); second, T_0, the number of time intervals used in fitting the auto-regressive model; and third, ϕ, the auto-regressive model's dimension (number of historical time intervals). T_0 cannot exceed the number of available intervals in the underlying dataset (see Table 2). According to the description of CE in [13], in order to estimate the coefficient matrices correctly, the following condition should be met $T_0 \geq k\phi$. As T_0 is limited, ϕ and k can not grow together. For example, let's consider the 4^{th} fold, choosing one week as the time interval gives 126 intervals in the training set. We can assign 100 for learning the auto-regressive model: $T_0 = 100$. If we want to build the auto-regressive model with looking at 4 weeks in the past ($\phi = 4$), then k can be at most 25. We know that this value of k is too small to learn good latent models in our dataset, it is desired to give higher values for ϕ and k and this is not possible as long as the previously mentioned condition should be met. Although CE builds a time-aware model, the limitation we explained here makes it inapplicable in such real-world datasets where the ratings frequency doesn't allow considering shorter time intervals.

CTR shows better performance in comparison with CF and CE as it utilizes the

Fold	Time interval		
	1 Day	1 Week	1 Month
1	327	48	11
2	508	74	17
3	692	100	23
4	873	126	29
5	1057	152	35

Table 2: Number of intervals in the dataset for each fold

content of the items in addition to the collaborative ratings. However, accounting for the concept-drift in user interest leads to the superiority of our method against all studied methods in all recorded metrics as shown in Figure 3.

5.5 User-specific vs Common Concept-drift Scores

In our last experiment, we analyze the advantage of computing the concept-drift score for each user individually. Therefore, we ran T-CTR with the following configurations: (a) **T-CTR**: where the concept-drift score is computed individually for each user as in Algorithm 1; (b)**T-CTR-*s***: where a common concept-drift score (s) is set for all users. We chose three values for s ranging from small to high: $s \in \{0.1, 0.5, 1\}$. As depicted in Figure 1, lower concept-drift scores give higher confidence weights for old ratings. When $s = 0.1$ for example, old ratings are lightly penalized and when $s = 1$ old ratings are strongly penalized.

The results are shown in Figure 4. The results of all evaluation metrics show that using individual concept-drift scores yields better results. To gain better

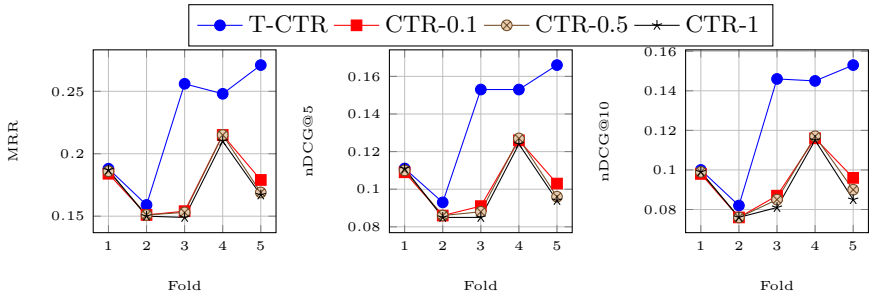

Fig. 4: Performance comparison. User-specific compared to common concept-drift score. These results show that computing the concept-score for each user individually leads to better recommendations.

understanding about the role of concept-drift score, we conducted qualitative analysis. We considered the 5^{th} fold and compared the performance of T-CTR-s for individual users across the different values of s. We found the following, when increasing s from 0.1 to 0.5, results improved for 87 users but worsened for 143, this means for 87 users $s = 0.5$ is a better choice than $s = 0.1$ and for 143 users it is the opposite case. Similar observation can be realized when moving to $s = 1$, compared to $s = 0.5$, the results got better for 109 users and worst for 134 users. An interesting question is whether those users which got better results when increasing s to 0.5 will also show better results for $s = 1$. We found that not all users who showed results improvement for $s = 0.5$ also showed improvement for $s = 1$, 46 of them got worst results and additional 68 users showed better results. This analysis supports our assumption that each user has an individual concept-drift score which does not fit necessarily other users. Above that, our suggested method to dynamically compute individual concept-drift scores leads to better results than assigning a common score for all users.

6 Conclusion and Future Work

In this paper, we introduced T-CTR, a time-aware approach for recommending textual items. Based on the heterogeneity of the items from user's historical ratings, we compute a personalized user-specific concept-drift score. Then, we use these scores to calculate confidence weights for known ratings. These weights control the ratings' contribution in fitting the CTR model. The take-away messages from this work is twofold: (a) in order to achieve realistic evaluation, it is essential to conduct time-aware evaluation method; and (b) as users have different concept-drift dynamics, concept-drift models should be computed for each user individually. The main aspect that we plan to investigate in our future work is to design a probabilistic model that allows *learning* the concept-drift score for each user instead of relying on the heuristic approach of calculating the average similarity of the user previous ratings.

References

1. Alzoghbi, A., Ayala, V.A.A., Fischer, P.M., Lausen, G.: Pubrec: Recommending publications based on publicly available meta-data. In: LWA Workshops: KDML, FGWM, IR, and FGDB, pp. 11–18 (2015)
2. Bansal, T., Belanger, D., McCallum, A.: Ask the gru: Multi-task learning for deep text recommendations. In: Proceedings of the 10th ACM Conference on Recommender Systems, RecSys '16. ACM (2016)
3. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. Journal of machine Learning research **3**(Jan), 993–1022 (2003)
4. Campos, P.G., Díez, F., Cantador, I.: Time-aware recommender systems: A comprehensive survey and analysis of existing evaluation protocols. User Modeling and User-Adapted Interaction **24**(1-2), 67–119 (2014)
5. Ding, Y., Li, X.: Time weight collaborative filtering. In: Proceedings of the 14th ACM international conference on Information and knowledge management, pp. 485–492. ACM (2005)
6. Gao, L., Wu, J., Zhou, C., Hu, Y.: Collaborative dynamic sparse topic regression with user profile evolution for item recommendation. In: AAAI Conference on Artificial Intelligence, pp. 1316–1322 (2017)
7. Hu, Y., Koren, Y., Volinsky, C.: Collaborative filtering for implicit feedback datasets. In: Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on, pp. 263–272. Ieee (2008)
8. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of ir techniques. ACM Trans. Inf. Syst. **20**(4) (2002)
9. Koren, Y.: Collaborative filtering with temporal dynamics. Communications of the ACM **53**(4), 89–97 (2010)
10. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)
11. Liu, N.N., Zhao, M., Xiang, E., Yang, Q.: Online evolutionary collaborative filtering. In: Proceedings of the Fourth ACM Conference on Recommender Systems, RecSys '10. ACM (2010)
12. Liu, X.: Modeling users' dynamic preference for personalized recommendation. In: Proceedings of the 24th International Conference on Artificial Intelligence, pp. 1785–1791 (2015)
13. Lu, Z., Pan, S.J., Li, Y., Jiang, J., Yang, Q.: Collaborative evolution for user profiling in recommender systems. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, pp. 3804–3810 (2016)
14. Matuszyk, P., Vinagre, J., Spiliopoulou, M., Jorge, A.M., Gama, J.: Forgetting techniques for stream-based matrix factorization in recommender systems. Knowledge and Information Systems **55**(2), 275–304 (2018)
15. Mnih, A., Salakhutdinov, R.R.: Probabilistic matrix factorization. In: Advances in neural information processing systems, pp. 1257–1264 (2008)
16. Pan, R., Zhou, Y., Cao, B., Liu, N.N., Lukose, R., Scholz, M., Yang, Q.: One-class collaborative filtering. In: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, ICDM '08, pp. 502–511. IEEE Computer Society, Washington, DC, USA (2008)
17. Vinagre, J., Jorge, A.M.: Forgetting mechanisms for scalable collaborative filtering. Journal of the Brazilian Computer Society **18**(4), 271–282 (2012)
18. Vinagre, J.a., Jorge, A.M., Gama, J.a.: An overview on the exploitation of time in collaborative filtering. Wiley Int. Rev. Data Min. and Knowl. Disc. **5**(5) (2015)

19. Wang, C., Blei, D.M.: Collaborative topic modeling for recommending scientific articles. In: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 448–456. ACM (2011)
20. Wang, H., Chen, B., Li, W.J.: Collaborative topic regression with social regularization for tag recommendation. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, pp. 2719–2725 (2013)
21. Wang, H., Wang, N., Yeung, D.Y.: Collaborative deep learning for recommender systems. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '15. ACM (2015)