On-the-Fly News Recommendation Using Sequential Patterns

Mozhgan Karimi
University of Antwerp
mozhgan.karimi@uantwerpen.be

Boris Cule
University of Antwerp
boris.cule@uantwerpen.be

Bart Goethals
University of Antwerp
bart.goethals@uantwerpen.be

ABSTRACT

The news recommendation problem poses a number of specific challenges that established recommendation techniques, successful in other settings, do not tackle adequately. For example, unlike in other domains, the relevance of news articles drops significantly over time, and the order in which users visit news articles matters greatly. Furthermore, in the context of breaking news, user interests can change rapidly, and there is a need to generate recommendations on-the-fly, taking into account recently published articles and the latest trends among users' preferences. To address these issues, we use a form of sequential pattern mining to generate up-to-date news recommendations on a click-by-click basis. In this approach, patterns are mined incrementally from the incoming clickstream so that new items and trends are considered. Our experimental evaluation demonstrates that our method compares favorably with existing techniques and outperforms them on a variety of metrics.

KEYWORDS

News Recommendation; Clickstream; Sessions; Sequential Patterns

1 INTRODUCTION

Each recommendation scenario has its own particular characteristics, which call for different approaches in designing recommender systems (RS). Thus far, numerous technical approaches have been proposed for generating recommendations, including *data mining* techniques, such as sequential pattern mining (SPM) methods [25]. However, specifically in the domain of news recommendation, SPM techniques have only seen limited use so far [9].

News is a peculiar application domain of RS for a number of reasons. For example, users mostly read news articles in the first two days after publication, after which the relevance of articles dramatically decreases [4]. In contrast, in other domains like music, users regularly listen to tracks from years ago. In addition, user interests are not completely stable on news websites. When trending news articles emerge, users often engage with news topics that do not fit their normal reading preferences. This change in user preference obviously occurs in other domains as well. For example, in the context of fashion, users sometimes adjust their preferences based on seasonal trends. However, such preference reversals are not comparable to sudden interest shifts that occur in the news domain, e.g., when a news article about a natural disaster is published.

It is, therefore, necessary to update news recommendation models immediately, which is not possible with most state-of-the-art recommendation approaches, such as those based on latent factors. Instead, a news recommendation model needs to allow for incremental updates based on every new click. In this paper, we propose a news recommendation approach based on the above-mentioned design goals. The algorithm mines patterns from ongoing user sessions *incrementally* and subsequently adds the patterns to a tree-based model. This model can be scanned efficiently for patterns that match a given user's current click session, and recommendations can, consequently, be generated in real-time based on the most recent information available.

In *7th International Workshop on News Recommendation and Analytics (INRA 2019), in conjunction with RecSys 2019, September 19, 2019, Copenhagen, Denmark.*

2 RELATED WORK

In the field of recommender systems, modeling temporal dynamics of user behavior and the evolution of items are challenging issues [5, 8, 13, 14], especially in the news domain [2, 10]. To address user interest drifts, Lu et al. use an attenuation function to prioritize news topics that users were most recently interested in [21], while Epure et al. model short-term interests using Markov processes without prior user models [3]. Other works rely mainly on traditional recommendation approaches, such as co-visitation patterns, and extend them with specialized user profiling strategies [17, 18, 22, 26].

Other approaches use contextual information, such as the time of day or the user's location [15, 19]. In the 2014 CLEF NewsREEL challenge [20], two teams used contextual information in the online task. The runner-up team used an agent-based approach that delegated recommendation requests to different algorithms based on the day of the week [19]. The third placed team implemented an Apriori-based rule-mining approach that utilizes click data and context features, such as the user's location [15].

Recently, deep learning techniques, specifically Recurrent Neural Networks (RNN), have also been used to model short-term user behavior, leading to promising results in session-based news recommendation scenarios [16, 24]. However, neural approaches are not always the most effective choice, as their long training times prevent them from immediately recommending new items, e.g., breaking news.

In contrast, traditional nearest neighbor schemes, which do not need to train extensive models, have recently been shown to be a strong baseline in the news domain and other recommendation scenarios where short-term interests are important [7]. Specifically, a session-based nearest neighbor method called V-SkNN, which prioritizes the most recent user clicks, was able to outperform a state-of-the-art neural approach on two news data sets [9]. The V-SkNN approach can, therefore, be considered as the current benchmark in news recommendation. We, consequently, include the approach for comparison in our empirical evaluation and follow the authors' evaluation protocol.

Another traditional approach used to generate recommendations in a number of domains is SPM [6, 25]. In the context of news, some papers employ frequent pattern mining approaches [1, 9] that consider each session as a single uninterrupted pattern. Our approach is somewhat similar to these strategies. However, our strategy exploits *all* possible subpatterns in each user session. We, thus, consider user click sessions in a more fine-grained way, which allows us to capture interest drifts over the course of a few clicks as well as the whole session context.

3 PROPOSED APPROACH

In this section, we first give a brief overview of the basics of sequential pattern mining. We then explain the proposed tree-based model and the recommendation mechanism.

3.1 Frequent Sequential Patterns

Sequential pattern mining is a well-established field in data mining. In the traditional setting, given a database of sequences, the goal is to discover which (sub)sequences occur often in the data [23]. Such sequences are called frequent sequential patterns.

Our setting is well-suited for SPM. The data consists of user sessions, each of which can be represented as a sequence of clicks, with its associated subsequences. Consequently, a frequent sequential pattern is an often encountered sequence of clicks. For example, the frequent sequential pattern {A, B, C} indicates that many users have first clicked on A, then on B, and, finally, on C. Therefore, if we know that the current user has clicked on A and then on B, we could recommend item C to the user. We require the patterns to be frequent, i.e., based on a high number of previous sessions, in order for the recommendations to be reliable.

3.2 Tree-based Model

Our proposed algorithm uses a tree-based model to store mined patterns, which allows us to easily update the recommendation model based on each incoming session. In the following, the tree model is described based on an example. Let two sessions be S_1 and S_2, where S_1 contains clicks on items {A, B} and S_2 includes clicks on items {A, C}.[1] To capture all (sub)sequences in the session, we first calculate the power set of the items in the session. For S_1 the power set would be {{A}, {B}, {A, B}}, and for S_2 it would be {{A}, {C}, {A, C}}. Next, we build the tree model based on the power sets. For all sets in the power set, we traverse the tree and increment the counter of the respective *last* node. For example, for the set {A, B}, the algorithm starts from the root and traverses through node A and then B. Reaching B, it increments the node's frequency counter by one. If there are items in the set for which no nodes exist in the tree, new nodes are created on-the-fly. This process is applied to all sets of the power set. Figure 1 shows the tree-based model after processing the two sessions from our example.

Since the number of subsets in a power set grows exponentially based on the session length, it is important to limit the power set size. Thus, we introduce an adjustable parameter called *sliding window* which limits the number of clicks in a pattern. If a user has a session longer than n clicks, the power set of the first n items is calculated. Then, the window slides by one, and the power set from the 2$^{\text{nd}}$ item to the n+1$^{\text{st}}$ item is calculated, and so on. Thus, we are even able to process patterns from longer sessions efficiently.

[1] We show the clicks of a user in the current session in set notation. However, the order of items in the set is important in our model.

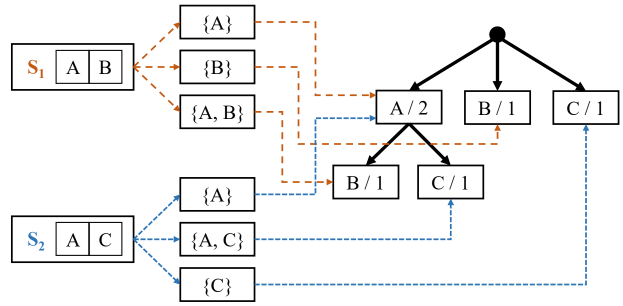


Figure 1: Creation of the Pattern Tree Model

Algorithm 1: Partial Power Set Creation

Data: $S^*_n = s^*_1, s^*_2, \ldots, s^*_n$: current (partial) user session
Used subroutines : $\mathcal{P}(X)$: creates the (full) power set of X
Result: P^*_n: partial power set
$S^*_{n-1} := S^*_n \setminus s^*_n$; ▷ `remove` s_n `(latest click)`
$P^*_n := \mathcal{P}(S_{n-1})$; ▷ `create power set of` S^*_{n-1}
foreach *Set* $Q^* \in P^*_n$ **do**
  $Q^* := Q^* \cup s_n$; ▷ `add` s_n `to each subset`
end
return P^*_n ;

3.3 Incremental Tree Creation

In reality, complete user sessions do not become available for processing right away, but instead click-by-click. However, waiting for a session to "finish", e.g., when a user has not clicked on an article for 20 minutes, is also not an option, as this would result in an inherently outdated model. The algorithm, therefore, needs to build the power set $\mathcal{P}(S)$ of each session S incrementally. To this end, when the n^{th} click is added to a session, the algorithm creates a partial power set P^*_n. The idea is that, in combination with the previously created partial power sets of that session, the complete power set will be assembled as $P^*_1 \cup P^*_2 \cup \ldots \cup P^*_n = \mathcal{P}(S)$. In this way, the subpatterns in the partial power sets can already be applied to the tree model while the user session is still ongoing.

The subroutine for creating a partial power set P^*_n based on an ongoing session S^*_n is given in Algorithm 1. We explain this subroutine with an example. Assuming that, so far, a user clicked on items A and B in an ongoing session, the algorithm will iteratively process the partial sessions $S^*_1 = \{A\}$ and $S^*_2 = \{A, B\}$. Thus, when processing the first session snapshot S^*_1, the algorithm would simply produce the empty set $\{\{\}\}$ and then add A, resulting in $P^*_1 = \{A\}$. This partial power set would then be applied to the pattern tree. When processing S^*_2 in the next step, the algorithm needs to produce a partial power set P^*_2 that does *not* overlap with the previously created P^*_1, as this information is already stored in the tree model. The algorithm will, thus, first build P^*_2 as $\{\{\}, \{A\}\}$ and then add B to each subset, to create P^*_2 as $\{\{B\}, \{A, B\}\}$. Together, these partial power sets P^*_1 and P^*_2 form the complete power set of the user's clicks session {A, B} as $P^*_1 \cup P^*_2 = \mathcal{P}(S) = \{\{A\}, \{B\}, \{A, B\}\}$.

3.4 Focusing on Recent Patterns

As mentioned, in news recommendation, it is paramount to keep track of recent user interests. To focus on more recent interest trends, while also reducing the volume of patterns in the model, we remove stale patterns from the tree based on a queue implementation of adjustable size (ω). As clicks occur, the respective session snapshots are added to this queue, and, once the queue is full, the oldest session snapshot is removed. The snapshot is then processed again to revert its effect on the pattern tree. That is, frequencies of (sub)sequences originating from this old session are decremented, and, in case a node's frequency is reduced to zero, the node is completely removed from the tree. Consequently, only the ω most recent clicks have an effect on the model, and the tree, thus, represents only the most recent interest trends among users.

3.5 Recommendation Mechanism

To generate recommendations, the score for each candidate item i from the set of all available items is calculated as follows:

$$score_i = \sum_{P \in \mathcal{P}(S)} conf(P), \quad \text{with}$$

$$conf(P) = \begin{cases} f(P \cup i)/f(P) & \text{if } f(P \cup i) \geq \sigma \ \wedge \ f(P \cup i)/f(P) \geq \kappa \\ 0 & \text{otherwise} \end{cases},$$

where $\mathcal{P}(S)$ is the power set of the current user session S, and $f(P)$ is a function that returns the frequency of a sequence P in the tree model. As can be seen from the formula, for each subset P of the power set $\mathcal{P}(S)$, the pattern tree is scanned for a matching pattern as well as a continuation of that pattern by the candidate item i, i.e., $P \cup i$. The confidence score $conf(.)$ for this tree branch is then calculated as the frequency of the pattern continuation $f(P \cup i)$ divided by the frequency of its parent pattern $f(P)$. Additionally, two threshold parameters are introduced to filter patterns of low potential usefulness. On the one hand, if the pattern continuation appears less than σ times in the data, it is not considered. On the other hand, if the confidence score is less than κ, the pattern is also not included in the calculation. This process is then continued for each tree branch that matches with a subset of the current session's power set. Finally, the items are ranked based on their accumulated confidence scores.

4 EXPERIMENTS

In this section, we elaborate on the design and results of an empirical evaluation of the proposed recommendation approach.[2]

4.1 Evaluation Setup

We design our experiments based on an open-source framework, called StreamingRec [9]. Its evaluation scheme simulates real-time recommendation by replaying time-stamped click log data sequentially as a stream. Consequently, algorithms receive the input data click-by-click as it occurred in reality, and they have to react to each click by making recommendations. Due to this stream-based protocol, algorithms can also learn incrementally from incoming clicks during the test phase and, thus, improve their models on-the-fly. The evaluation protocol used in our experiments is thus comparable to the *replay* protocol of the CLEF NewsREEL challenge [20].

To demonstrate the effectiveness of our proposed approach, we compare it with a variety of algorithms implemented in the StreamingRec framework including a popularity baseline, an item-based co-occurrence scheme, and a session-based nearest neighbor method. The popularity baseline (Pop.) works by simply recommending the articles that were clicked most often in the data set. The recently popular algorithm (R. Pop.) ranks candidates by their popularity within the recent past (20 minutes in our setting). The co-occurrence method (Co-Occ.), on the other hand, selects recommendation candidates based on how often they occurred together in other sessions with items from the target user's session. The nearest neighbor scheme (V-SkNN) performed best in previous evaluations with the StreamingRec framework [9]. It scores candidate items from neighborhood sessions based on the overlap between the target session and the neighboring session, with a boost in the similarity function for recent clicks in the target session. We compared our proposed approach with further algorithms from the StreamingRec framework. However, due to space constraints, we only report the most representative results. For our proposed approach, we implemented the following variations:

- **Seq**: Recommendation mechanism described in Section 3.5.
- **Seq$_r$**: Similar to Seq but also takes the recency of clicks into consideration as described in Section 3.4.
- **Seq$_p$**: The same method as Seq except for the candidate scoring method. This approach penalizes candidate items generated from longer patterns. To this end, instead of dividing by $f(P)$ in the confidence score formula, we accumulate the support values of all parent nodes and use it as a divisor.
- **Seq$_{pr}$** Combines the recency and penalization varieties.

We evaluate our approach on two real-world data sets from Plista and Outbrain. The Plista data set contains German news published by 12 publishers during June 2013 [11, 12]. We chose a medium-sized publisher (ID: 418) with about 1.1 million clicks and more than 220 000 users. From the Outbrain data set[3], which is based on US news from June 2016, we also chose a publisher (ID: 43) with roughly 1.1 million clicks and around 280 000 users. We employ a temporal split into training and test data by 70% and 30%, respectively. Additionally, we tuned the parameters for each algorithm on a separate validation set based on 10% of the training data. The best parameter settings for Seq$_r$ were found to be a recency queue size of $\omega = 9000$, support threshold $\sigma = 1$, and confidence threshold $\kappa = 0.03$ for the Plista data set; for the Outbrain data set, $\omega = 17000$, $\sigma = 1$, and $\kappa = 0.09$. Moreover, we consider an idle time of 20 minutes per user to split the clickstream into user sessions, and we use a list cut-off for the metrics at 10.

4.2 Results

F1 Performance. Table 1 shows detailed performance results on the Plista and Outbrain data sets. In terms of F1, the baseline algorithms such as Random and Most Popular perform poorly as they do not consider new items and the recency of the clickstream. On the other hand, the Co-Occurrence and Recently Popular methods have relatively reasonable F1 performance, as they utilize the session context. The proposed Seq$_r$ approach is the best-performing

[2]The code for our proposed approach is included as supplementary material, which can be found here: https://github.com/mozhgank/StreamingRec

[3]https://www.kaggle.com/c/outbrain-click-prediction/data

Table 1: Performance Comparison. MRR = Mean Reciprocal Rank, RI = Number of Recommended Items, T = Time. Metrics use a list cut-off at 10. Best results are indicated in bold. All pairwise differences w.r.t. F1 and MRR are significant.

	Plista				Outbrain			
	F1	MRR	RI	T(ms)	F1	MRR	RI	T(ms)
Random	0.002	0.004	**1088**	0.044	0.002	0.003	**1564**	0.053
Pop.	0.005	0.007	12	0.347	0.003	0.007	11	0.192
R. Pop.	0.133	0.216	278	0.168	0.117	0.209	156	0.192
Co-Occ.	0.137	0.198	686	0.084	0.140	0.296	1064	0.120
V-SkNN	0.163	**0.361**	572	2.164	0.158	**0.344**	1178	5.716
Seq	0.159	0.260	501	0.022	0.195	0.278	1079	0.116
Seq_r	**0.178**	0.286	387	**0.011**	**0.212**	0.296	962	**0.041**
Seq_p	0.141	0.296	505	0.038	0.138	0.300	1077	0.209
Seq_{pr}	0.157	0.325	388	0.017	0.151	0.326	955	0.075

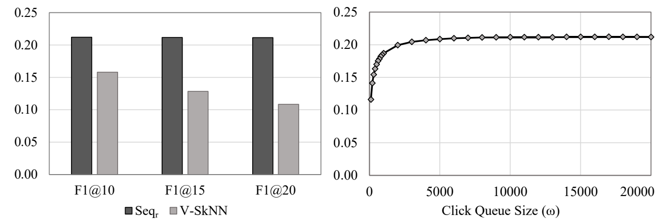
strategy for both data sets. The improvement in terms of F1 in comparison with V-SkNN—the best-performing strategy in previous experiments [9]—amounts to 34 and 9 percent on the Outbrain and Plista data sets, respectively. Moreover, comparing the different varieties of the proposed approach shows that the recency-aware methods always exhibit better F1 performance.

MRR Performance and Aggregate Diversity. Looking at the Mean Reciprocal Rank measure, the best performing algorithm is V-SkNN, due to its prioritization of items closer to the current click of the user in the session for similarity calculation. All variations of our approach are, however, not far behind V-SkNN in terms of MRR. With respect to aggregate diversity, which measures the number of unique items recommended by each algorithm (RI), the Random strategy scores highest, which is not surprising. However, most of the more advanced algorithms, including Seq, are also able to recommend from a large pool of items.

Runtimes. We also report computation times for each algorithm, i.e., the time it takes, on average, to generate a recommendation list. Here, we can observe that the best method in terms of F1 (Seq_r) also performs fastest. In addition, as expected, methods that consider recency aspects by filtering "old clicks" are faster compared to methods that build their model based on the whole clickstream history. In contrast, the session-based nearest neighbor method (V-SkNN) is not as efficient in creating recommendations as the proposed approach. One possible reason for this could be the large number of sessions that have to be compared with the current session, while the proposed method, in contrast, only has to scan a few branches of the pattern tree.

Effect of Window Size. As already mentioned, our approach features a sliding window constraint that limits the length of patterns to prevent exponential growth of the pattern tree. We experimented with different window sizes to observe the effect on accuracy. As expected, longer window sizes lead to better performance, but also slower processing times. However, at a certain point, performance actually decreases with longer window sizes, indicating that after a certain number of user clicks, item relations are much less meaningful. The experiments revealed that a window size of 5 was a good choice on both data sets.

Performance w.r.t. List Length. As a more detailed analysis of the effectiveness of the proposed approach, we evaluated its F1 performance with respect to different cut-off thresholds for the



(a) F1 Results w.r.t. List Length **(b) Effect of Recency on F1@10**

Figure 2: Performance Results for Outbrain data set

recommendation list, i.e., F1@10, 15, and 20. This way, we can estimate more clearly how useful the recommendations would be in different application scenarios that require, for example, short lists in mobile apps or longer lists on news websites optimized for desktop use. The results for the Outbrain data set show that the F1 performance of Seq_r remains steady around 0.21 when the list cut-off increases, while V-SkNN's performance drops (see Figure 2a). Thus, compared to V-SkNN, which can only be applied where short recommendation lists are needed, the proposed Seq_r approach shows promising results for a wider range of list lengths.

Effect of Recency. Finally, we seek to understand the effect of recency on accuracy. Thus, we investigated different queue size settings (ω) for the best method proposed in this work, namely Seq_r. As shown in Figure 2b, the larger the queue size, the better the performance in terms of F1 becomes. For example, for the Outbrain data set, when only patterns based on the last 500 clicks are considered, the algorithm achieves an F1 of 0.17, while 2000 clicks already produce an F1 of 0.20. However, after a certain point, performance decreases again. Interestingly, our proposed approach performs better than V-SkNN even when we only work with the 400 most recent clicks in the stream. In contrast, V-SkNN needs to consider 4 500 neighboring sessions to achieve its most optimized F1 performance, which still falls behind our approach, highlighting the suitability of a pattern-based strategy for the news domain.

5 CONCLUSION

The principal aim of our work was to recommend up-to-date news articles that fit user interests in a session-based scheme. To this end, we implemented an algorithm based on sequential pattern mining that extracts patterns incrementally from a continuous clickstream. Our empirical evaluation revealed that the approach's ability to capture temporal dynamics in user behavior led to improved accuracy compared to state-of-the-art news recommendation approaches. In addition, the experiments showed that extracting patterns based mainly on the most recent clicks in the data stream is not only more efficient but also leads to higher accuracy.

In the future, we plan to extend our session-based approach by also considering long-term user click sequences to create personalized news recommendations. Additionally, more complex weighting functions based on the position of clicks within sessions are planned. Lastly, we aim to explore the capabilities of our approach in other evaluation settings and domains as well as in online studies.

REFERENCES

[1] Pieter Bons, Nick Evans, Peter Kampstra, and Timo van Kessel. 2017. A News Recommender Engine with a Killer Sequence. In *Working Notes of the 8th Conference and Labs of the Evaluation Forum (CLEF '17)*.

[2] Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google News Personalization: Scalable Online Collaborative Filtering. In *Proceedings of the 16th International Conference on World Wide Web (WWW '07)*. 271–280.

[3] Elena Viorica Epure, Benjamin Kille, Jon Espen Ingvaldsen, Rebecca Deneckere, Camille Salinesi, and Sahin Albayrak. 2017. Recommending Personalized News in Short User Sessions. In *Proceedings of the Eleventh ACM Conference on Recommender Systems (RecSys '17)*. 121–129.

[4] Jon Atle Gulla, Cristina Marco, Arne Dag Fidjestøl, Jon Espen Ingvaldsen, and Özlem Özgöbek. 2016. The Intricacies of Time in News Recommendation. In *Extended Proceedings of the 24th Conference on User Modeling, Adaptation and Personalisation (UMAP '16)*.

[5] Wenxing Hong, Lei Li, and Tao Li. 2012. Product recommendation with temporal dynamics. *Expert Systems with Applications* 39, 16 (2012), 12398–12406.

[6] Cheng-Lung Huang and Wei-Liang Huang. 2009. Handling sequential pattern decay: Developing a two-stage collaborative recommender system. *Electronic Commerce Research and Applications* 8, 3 (2009), 117–129.

[7] Dietmar Jannach and Malte Ludewig. 2017. When Recurrent Neural Networks Meet the Neighborhood for Session-Based Recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems (RecSys '17)*. 306–310.

[8] Olivier Jeunen, Koen Verstrepen, and Bart Goethals. 2018. Fair Offline Evaluation Methodologies for Implicit-Feedback Recommender Systems with MNAR Data. In *Proceedings of the 2018 Workshop on Offline Evaluation for Recommender Systems (REVEAL '18)*.

[9] Michael Jugovac, Dietmar Jannach, and Mozhgan Karimi. 2018. StreamingRec: A Framework for Benchmarking Stream-based News Recommenders. In *Proceedings of the 12th Conference on Recommender Systems (RecSys '18)*. 269–273.

[10] Mozhgan Karimi, Dietmar Jannach, and Michael Jugovac. 2018. News recommender systems – Survey and roads ahead. *Information Processing & Management* 54, 6 (2018), 1203–1227.

[11] Benjamin Kille, Frank Hopfgartner, Torben Brodt, and Tobias Heintz. 2013. The Plista Dataset. In *Proceedings of the 2013 International News Recommender Systems Workshop and Challenge (NRS '13)*. 16–23.

[12] Benjamin Kille, Andreas Lommatzsch, and Torben Brodt. 2015. News Recommendation in Real-Time. In *Proceedings of the 2010 Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD '10)*. 149–180.

[13] Noam Koenigstein, Gideon Dror, and Yehuda Koren. 2011. Yahoo! Music Recommendations: Modeling Music Ratings with Temporal Dynamics and Item Taxonomy. In *Proceedings of the Fifth ACM Conference on Recommender Systems (RecSys '11)*. 165–172.

[14] Yehuda Koren. 2009. Collaborative Filtering with Temporal Dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '09)*. 447–456.

[15] Jaroslav Kuchar and Tomás Kliegr. 2014. InBeat: Recommender System as a Service. In *Working Notes of the 5th Conference and Labs of the Evaluation Forum (CLEF '14)*.

[16] Vaibhav Kumar, Dhruv Khattar, Shashank Gupta, Manish Gupta, and Vasudeva Varma. 2017. User Profiling Based Deep Neural Network for Temporal News Recommendation. In *Proceedings of the 2017 IEEE International Conference on Data Mining Workshops (ICDMW '17)*. 765–772.

[17] H.J. Lee and Sung Joo Park. 2007. MONERS: A news recommender for the mobile web. *Expert Systems with Applications* 32, 1 (2007), 143–150.

[18] Lei Li, Li Zheng, Fan Yang, and Tao Li. 2014. Modeling and broadening temporal user interest in personalized news recommendation. *Expert Systems with Applications* 41, 7 (2014), 3168–3177.

[19] Andreas Lommatzsch. 2014. Real-Time News Recommendation Using Context-Aware Ensembles. In *Proceedings of the 36th European Conference on Information Retrieval (ECIR '14)*. 51–62.

[20] Andreas Lommatzsch, Benjamin Kille, Frank Hopfgartner, Martha Larson, Torben Brodt, Jonas Seiler, and Özlem Özgöbek. 2017. CLEF 2017 NewsREEL Overview: A Stream-Based Recommender Task for Evaluation and Education. In *Proceedings of the 8th Conference and Labs of the Evaluation Forum (CLEF '17)*. 239–254.

[21] Meilian Lu, Zhen Qin, Yiming Cao, Zhichao Liu, and Mengxing Wang. 2014. Scalable news recommendation using multi-dimensional similarity and Jaccard–Kmeans clustering. *Journal of Systems and Software* 95 (2014), 242–251.

[22] Nikhil Muralidhar, Huzefa Rangwala, and Eui-Hong Sam Han. 2015. Recommending Temporally Relevant News Content from Implicit Feedback Data. In *Proceedings of the 27th International Conference on Tools with Artificial Intelligence (ICTAI '15)*. 689–696.

[23] Ramakrishnan Srikant and Rakesh Agrawal. 1996. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology (EDBT '96)*. 1–17.

[24] Lemei Zhang, Peng Liu, and Jon Atle Gulla. 2018. A Deep Joint Network for Session-based News Recommendations with Contextual Augmentation. In *Proceedings of the 29th Conference on Hypertext and Social Media (HT '18)*. 201–209.

[25] Baoyao Zhou, Siu Cheung Hui, and Kuiyu Chang. 2004. An intelligent recommender system using sequential Web access patterns. In *Proceedings of the 2004 IEEE Conference on Cybernetics and Intelligent Systems (CIS '04)*. 393–398.

[26] Zhiliang Zhu, Deyang Li, Jie Liang, Guoqi Liu, and Hai Yu. 2018. A Dynamic Personalized News Recommendation System Based on BAP User Profiling Method. *IEEE Access* 6 (2018), 41068–41078.