FlowRec: Prototyping Session-Based Recommender Systems in Streaming Mode

Dimitris Paraschakis(✉) and Bengt J. Nilsson

Malmö University, Nordenskiöldsgatan 1, 211 19 Malmö, Sweden
{dimitris.paraschakis,bengt.nilsson.TS}@mau.se

Abstract. Despite the increasing interest towards session-based and streaming recommender systems, there is still a lack of publicly available evaluation frameworks supporting both these paradigms. To address the gap, we propose `FlowRec` — an extension of the streaming framework `Scikit-Multiflow`, which opens plentiful possibilities for prototyping recommender systems operating on sessionized data streams, thanks to the underlying collection of incremental learners and support for real-time performance tracking. We describe the extended functionalities of the adapted prequential evaluation protocol, and develop a competitive recommendation algorithm on top of `Scikit-Multiflow`'s implementation of a Hoeffding Tree. We compare our algorithm to other known baselines for the next-item prediction task across three different domains.

Keywords: Streaming recommendations · Session-based recommendations · Prequential evaluation · Online learning · Hoeffding Tree

1 Introduction

In the past few years, the RecSys community has witnessed a paradigm shift from the traditional matrix completion problem to sequential *session-based* recommendations [9,15]. The latter approach is dominated by neural methods that are often evaluated in an online manner, i.e. when the events of a session are sequentially revealed and predicted one-by-one. However, these systems are still trained in batches on large chunks of recorded data [7,15,19]. To better approximate real-world scenarios with severe cold-start and concept drifts, *streaming* recommender systems [3,18,20] have been designed for incremental online learning from continuous data streams in the context of limited memory/runtime, and anytime prediction [17]. However, most of them address the conventional rather than session-based recommendation problem [6]. Bridging the gap between session-based and streaming recommender systems has been recently attempted [6,9], marking an emerging research direction of a high practical value.

H. W. Lauw et al. (Eds.): PAKDD 2020, LNAI 12084, pp. 65–77, 2020.
https://doi.org/10.1007/978-3-030-47426-3_6

Presently, only a few publicly available benchmarking frameworks for streaming recommendations exist. Some of them have been designed for a specific application domain [9,16], while others lack native support for session data [5,11]. `Scikit-Multiflow` [14] has recently been released as a general-purpose Python framework for stream mining, offering a variety of stream learners, change detectors, and evaluation methods. To facilitate the research on streaming session-based recommendations, we propose `FlowRec`[1] — an extension of `Scikit-Multiflow` for rapid prototyping of recommender systems. The proposed framework currently contains several stream-oriented recommenders and metrics for prequential evaluation. Additionally, we demonstrate a principled way of exposing a recommendation interface to an underlying stream learner class of `Scikit-Multiflow` (namely, a Hoeffding Tree). We show that the resulting recommender system has remarkable performance against established baselines. `FlowRec`'s functionality is detailed in the next section.

2 FlowRec

The framework consists of three main entities: a stream, an evaluator, and a model. This section describes the interplay between these entities.

2.1 Problem Setting

Consider a stream D of (overlapping) user sessions $S = s_1, \ldots, s_{|S|}$ (Fig. 1). A session represents an ordered sequence of events of the form (X, y), where X is a feature vector describing the context for item y. As a bare minimum, X contains the session identifier for the item. Other common features are timestamp and event type (e.g. click, purchase, etc.). The scope of our study is limited to the context of collaborative filtering, which relaxes the assumption of item metadata in feature vectors (technically, any feature can be encoded as a part of X).

At each time step $t = 1, \ldots, T$, the stream provides a sample (X, y). Based on the information in X, the model is asked to generate a list of N predictions $\hat{Y} = (\hat{y}_1, \ldots, \hat{y}_N)$ in an attempt to correctly guess the hidden item y. This corresponds to the *next-item prediction task* in the RecSys literature [15]. In practice, only certain features of X are retained for the prediction part, such as the current session identifier, and possibly the timestamp of the event. After the prediction, the entire feature vector X together with the label y are revealed to the model, allowing it to make an incremental update. This iterative, supervised 'test-then-train' methodology is known as *prequential evaluation* [20] (Fig. 2).

In our framework, each processed sample (X, y) is added to the sliding window of the last n observations, which we refer to as *observation window* (depicted as the green box in Fig. 1). Although its use is not required in the ordinary stream learning, it can ease the development of session-based models by providing a snapshot of the recent session data on demand. The size of the window must be chosen in consideration of the system's memory and runtime constraints.

[1] https://git.io/flowrec.

2.2 Metrics

FlowRec implements several evaluation metrics, two of which are commonly used for next-item prediction [6,7,13,19], namely *recall* (a.k.a. hitrate) and *mean reciprocal rank* (MRR). Recall measures the average number of successful predictions, whereas MRR measures their average reciprocal ranking, i.e.:

$$Recall@N = \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{N}\mathbb{1}(y_t = \hat{y}_i) \qquad MRR@N = \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{N}\frac{1}{i}\mathbb{1}(y_t = \hat{y}_i) \quad (1)$$

The framework keeps two sets of measurements: (a) *global*, where the running average of each metric is calculated from all the past data; and (b) *sliding*, where the average is taken over a sliding window of recent events that we call the *evaluation window* (purple box in Fig. 1). The sizes of evaluation and observation windows are user-adjustable, offering flexibility in simulation setups.

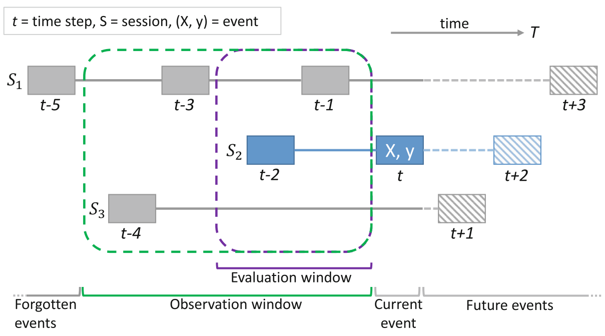


Fig. 1. Streaming sessions (Color figure online)

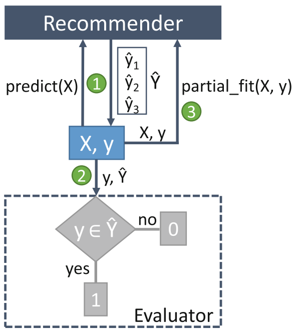


Fig. 2. Prequential protocol

2.3 Prequential Evaluation

The basic workflow for measuring recall using prequential evaluation is presented in Algorithm 1. The complete functionality of Scikit-Multiflow's prequential evaluator is provided in its official documentation[2].

FlowRec introduces the following additional parameters[3] for the evaluator:

- Indices of data columns holding session, timestamp, and event type identifiers. The last two columns are optional, and allow for time-aware and event-specific training and/or evaluation.
- Stream-related configurations, such as the size of the observation window, and the number of events to skip from the start of the stream.

2 https://scikit-multiflow.github.io/scikit-multiflow/documentation.html.
3 https://flowrec.readthedocs.io/en/latest/eval_parameters.html.

– Recommendation-specific settings, such as the size of the recommendation list, and the event types that trigger recommendation requests. There are also flags for enabling/disabling *reminders* and *repeated* recommendations. Reminders are recommendations of items that were *visited* earlier by the user, whereas repeated recommendations are those that were already *given* earlier to the user.

Algorithm 1. Basic prequential protocol for measuring recall

```
Input: $D$: data stream, $N$: recommendation cutoff, $n_{keep}$: size of the observation window, $M$: set of recommendation models
Output: Recall@$N$ of each model
 1: $W \leftarrow create\_queue(n_{keep})$          ▷ observation window of $n_{keep}$ latest events
 2: $r_m \leftarrow 0, \forall m \in M$          ▷ reward counter
 3: $n \leftarrow 0$          ▷ evaluation counter
 4: while $D.has\_more\_samples()$ do
 5:     $X, y \leftarrow D.next\_sample()$
 6:     if $X.session \in W.sessions$ then
 7:         for all $m \in M$ do
 8:             $\hat{Y} \leftarrow m.predict(X)$
 9:             if $y \in top\_N(\hat{Y})$ then
10:                 $r_m \leftarrow r_m + 1$
11:         $n \leftarrow n + 1$
12:     for all $m \in M$ do
13:         $m.partial\_fit(X, y)$
14:     $W.add(X, y)$
15: return $r_m/n, \forall m \in M$
```

The size of the evaluation window (as well as other settings) are covered by the original parameter list of the EvaluatePrequential class in `Scikit-Multiflow`. Note that the stream provides samples in the order as they appear in the dataset. The last column of the dataset should always contain the item identifiers.

3 Prototyping

Prototyping incremental session-based recommendation models in `FlowRec` is straightforward. First, a streaming model is built by extending the BaseSKMObject class of `Scikit-Multiflow` with the appropriate mixin. It is natural to treat the recommendation task as a multi-class classification problem, where each class corresponds to an item. Hence, the suitable mixin for this type of problems is ClassifierMixin[4]. What remains is to implement the following abstract methods:

[4] https://scikit-multiflow.github.io/scikit-multiflow/user-guide.core-concepts.architecture.html.

partial_fit(X, y) — Incrementally train a stream model.
predict(X) — Generate top-N predictions for the target's class.
predict_proba(X) — Calculate the probabilities of a sample pertaining to each of the available classes (implementing this method is optional).

Every stream model developed in `FlowRec` has access to useful shared resources, such as observation window, current session vector, and item catalog.

3.1 Session-Based Streaming Models

Presently, `FlowRec` contains the following streaming models for session-based recommendations:

Rule-Based Models. The first three models are rule-based recommenders that capture one-to-one relationships between items. These methods rely on the very last item of a session to make their predictions for the next item. Despite their simplicity, rule-based models have proven surprisingly effective in the domains of music and e-commerce [13], and have very low computational complexity. We briefly outline these methods below (refer to [13] for details).

Association Rules (AR). The rules are derived from co-occurrences of two items in a session (e.g. 'those-who-bought-also-bought'). A co-occurrence forms rules in both directions, i.e. $y_t \leftrightarrow y_{t'}, t \neq t'$.

Markov Chains (MC). The rules are derived from a first-order Markov Chain, describing the transition probability between items that appear in two *contiguous* events in a session, i.e. $y_t \rightarrow y_{t+1}$.

Sequential Rules (SR). The rules are derived from sequential patterns between two items in a session, but not necessarily in successive events, i.e. $y_t \rightarrow y_{t'}, t' > t$. The sequential association is assigned the weight $1/(t' - t)$.

In `FlowRec`, the above algorithms can be made event-specific by providing event_column_index. For instance, predictors of type *purchase* $\leftrightarrow$ *purchase*, *click* $\rightarrow$ *purchase*, etc. can be useful in e-commerce applications as components of an ensemble recommender [2].

Session *k*NN. This is a specialized version of the k-Nearest Neighbors (kNN) algorithm that operates on session data. It is a strong baseline with performance comparable to that of certain deep neural methods [8,13]. Being model-free, the algorithm is incremental by nature. S-kNN recommends items from other user sessions that are similar to the current session (a.k.a. neighbors). Given the active session S, the score of the candidate item $\hat{y}$ is calculated as follows:

$$score(\hat{y} \mid \hat{y} \notin S) = \sum_{S' \in \mathcal{N}(S)} sim(S, S') \cdot \mathbb{1}(\hat{y} \in S') \tag{2}$$

where $\mathcal{N}(S)$ is the k-sized neighborhood of session S, and $sim(S, S')$ is a measure of similarity between two sessions.

`FlowRec` implements Cosine, Jaccard, Dice, and Tanimoto similarity. In the future, we plan to implement other kNN variants described in [13].

BEER[TS]. This is a bandit ensemble designed for streaming recommendations, which employs Thompson Sampling for the model selection. In [2], event-specific rule-based models were used as behavioral components of the ensemble. In `FlowRec`, any model implementing the predict_proba(X) method can be added as a component of BEER[TS]. For each recommendation slot, the ensemble picks a model m with the highest sample $\theta_m \sim Beta(\alpha_m + 1, \beta_m + 1)$, where α_m is the number of past successes, and β_m is the number of past failures. The method is adaptable to non-stationary data (e.g. occurring due to concept drift), which is achieved via exploration-exploitation. In addition, BEER[TS] supports component splitting into (relatively) stationary partitions (see [2] for details), which is achieved in `FlowRec` by setting the boundaries for probabilities returned by the predict_proba(X) method. Further, the components of the ensemble can complement each other in case of poor coverage, which helps to attack sparsity and cold-start issues.

Popularity Baseline. This is the traditional baseline that outputs the top-N most popular items in descending order.

The above methods can be incrementally trained either on the global scale, or within the observation window. The latter option acts as a forgetting mechanism for older data, which aids model scalability and adaptability to recent trends.

3.2 Hoeffding Tree Wrapper

Prototyping streaming session-based recommenders can be facilitated by employing the rich collection of incremental algorithms offered by `Scikit-Multiflow`, including Bayesian methods, lazy learners, ensembles, neural networks, tree-based methods, and more [14]. Utilizing any of these methods for recommendation tasks is achieved via a *wrapper*, which is a middle layer that handles the inputs and the outputs of an underlying learner. Some of the common tasks performed by wrappers include:

- transforming a sample to the desired input format accepted by a learner.
- calling the predict_proba(X) method of a learner, and manipulating its return values to generate top-N recommendations.

Using the above approach, we develop a recommender system by 'wrapping' the HoeffdingTree classifier provided by `Scikit-Multiflow`.

Hoeffding Tree (HT). Also known as a Very Fast Decision Tree (VFDT) [4], a HT is an incremental, anytime decision tree inducer designed for learning from data streams. Its key idea lies in the fact that only a small subset of samples passing through a node may be sufficient for deciding on the split attribute. For estimating the minimum number of samples needed, the method employs the Hoeffding bound, which offers sound theoretical guarantees of performance (see [4] for details), asymptotically comparable to that of a batch decision tree.

HT Wrapper Architecture. The `Scikit-Multiflow`'s implementation of a HT supports Naive Bayes prediction at the leaves of the tree, and the possibility of assigning a weight to each fitted sample. We take advantage of both capabilities in the proposed HT recommender. The core idea of our algorithm is to encode all item-to-item associations employed by rule-based methods (see above) in a unified learner. This allows HT to capture both sequential and co-occurrence patterns in a session. The idea is conceptually similar to the BEER[TS] framework [2], but instead of treating between-items associations as separate predictors explored by a bandit, we fit them to a single decision tree classifier using a specific weighting scheme, as explained below. We hence formulate the recommendation task as a multi-class classification problem. Due to the nature of the problem, HT is reduced to a *decision stump*, whose nodes represent input items, and the leaves contain item predictions obtained via Naive Bayes classification.

Training the Model. The incremental training of a HT in `Scikit-Multiflow` can be done by calling the method partial_fit(X, y, sample_weight). The first two parameters specify the input feature vector and the output label, respectively, with an associated (optional) sample weight. We use these parameters to encode the sequential relation between two items, by letting the feature vector contain the antecedent item and the label represent the consequent item. For the ease of notation, we use the item in place of a feature vector in Algorithm 2 (y or y', lines 7 and 8). Internally, each antecedent is represented as a node of the tree.

Algorithm 2. HT Wrapper training procedure

```
Input: $(X, y)$: sample from the stream; $w_{MC} \geq 1$: importance weight for Markov
    Chain sequences; $w_{inv} \in [0, 1]$: importance weight for inverse sequences.
1: $S \leftarrow (y'_1, \ldots, y'_{|S|})$    ▷ item vector (in time order) for the session id encoded in $X$
2: for $i \leftarrow 1, \ldots, |S|$ do
3:     if $i = |S|$ then
4:         $w \leftarrow w_{MC}$    ▷ weight for the ‘next-item’ sequence
5:     else
6:         $w \leftarrow 1/(|S| - i + 1)$    ▷ weight for a non-contiguous sequence
7:     ht.partial_fit($y'_i, y, w$)    ▷ fit observed sequence
8:     ht.partial_fit($y, y'_i, w \cdot w_{inv}$)    ▷ fit inverse (unobserved) sequence
```

After fetching the current session vector S from the observation window, the wrapper learns all sequential patterns involving item y by fitting a series of samples $(y'_i, y, w), \forall i = 1, \ldots, |S|$, where the sample weight is inversely proportional to the distance between two items. Clearly, these fits utilize the same patterns as captured in Sequential Rules (SR) described above. Among these patterns, $y_{|S|} \rightarrow y$ pertains to the Markov Chain (MC). The corresponding sample, $(y'_{|S|}, y, w_{MC})$, uses a separate weight reflecting the perceived importance of the 'next-item' sequence. Finally, encoding co-occurrence patterns captured by Association Rules (AR) is achieved via a series of the so-called *inverse fits*, i.e. $(y, y'_i, w \cdot w_{inv}), \forall i = 1, \ldots, |S|$, which complete the bidirectional associations. The fixed weight $w_{inv} \in [0, 1]$ is used to inform the influence of inverse (hence unobserved) sequential patterns on the classification.

Making Predictions. Unlike rule-based methods, where the predictions are made solely on the basis of the latest item in a session, our HT wrapper makes predictions in an ensemble-like manner by combining the responses of all the relevant nodes of the tree. The prediction procedure is detailed in Algorithm 3.

Algorithm 3. HT Wrapper prediction procedure

```
Input: $X$: feature vector containing session id; $w_{MC} \geq 1$: importance weight for
    Markov Chain sequences; $w_{inv} \in [0,1]$: importance weight for inverse sequences;
    $N$: recommendation cutoff
Output: $\hat{Y}$: top-$N$ recommendations
 1: $S \leftarrow (y'_1, \ldots, y'_{|S|})$        ▷ fetch item vector for the current session id encoded in $X$
 2: $P \leftarrow (\mathbb{P}(y_1), \ldots, \mathbb{P}(y_{|P|}))$, set $\mathbb{P}(y_i) \leftarrow 0, \forall i, \ldots, |P|$        ▷ init class probabilities
 3: for $i \leftarrow 1, \ldots, |S|$ do
 4:     $P_i \leftarrow$ ht.predict_proba$(y'_i)$        ▷ predict class probabilities from item $y'_i$
 5:     if $i = |S|$ then
 6:         $P_i \leftarrow P_i \cdot w_{MC}$
 7:     else
 8:         $P_i \leftarrow P_i / (|S| - i + 1)$
 9:     $P \leftarrow P + P_i$
10: return $\hat{Y} \leftarrow (y_1, \ldots, y_N), \forall \mathbb{P}(y_i) \in$ top-$N(P)$
```

For each item y'_i in the current session vector, the wrapper calls the predict_proba(y'_i) method of a HT. This method returns a class probability vector that expresses the likelihood of each candidate item to follow item y'_i. The probability vectors P_i are then weighted with the recency of item y'_i. Predictions obtained from the most recent item, $y'_{|S|}$, receive the highest weight specified by w_{MC}. All weighted probability vectors are then added to produce the final scores for the candidate items, top-N of which are returned.

4 Simulation Results

4.1 Datasets

We use public datasets containing sessionized browsing logs. The datasets originate from three recommendation contests representing news, travel, and e-commerce domains. They are summarized in Table 1.

Table 1. Datasets summary (1M events each)

Dataset	Contest	Domain (Action)	Items	Sessions	Avg. session size
Clef	NewsReel'15 [10][a]	News (impressions)	109	305703	3.27 events
Yoochoose	RecSys'15 [1]	E-commerce (clicks)	21300	255166	3.92 events
Trivago	RecSys'19 [12]	Travel (clickouts)	243714	521677	1.92 events

[a]We use the subset of the dataset provided in [13].

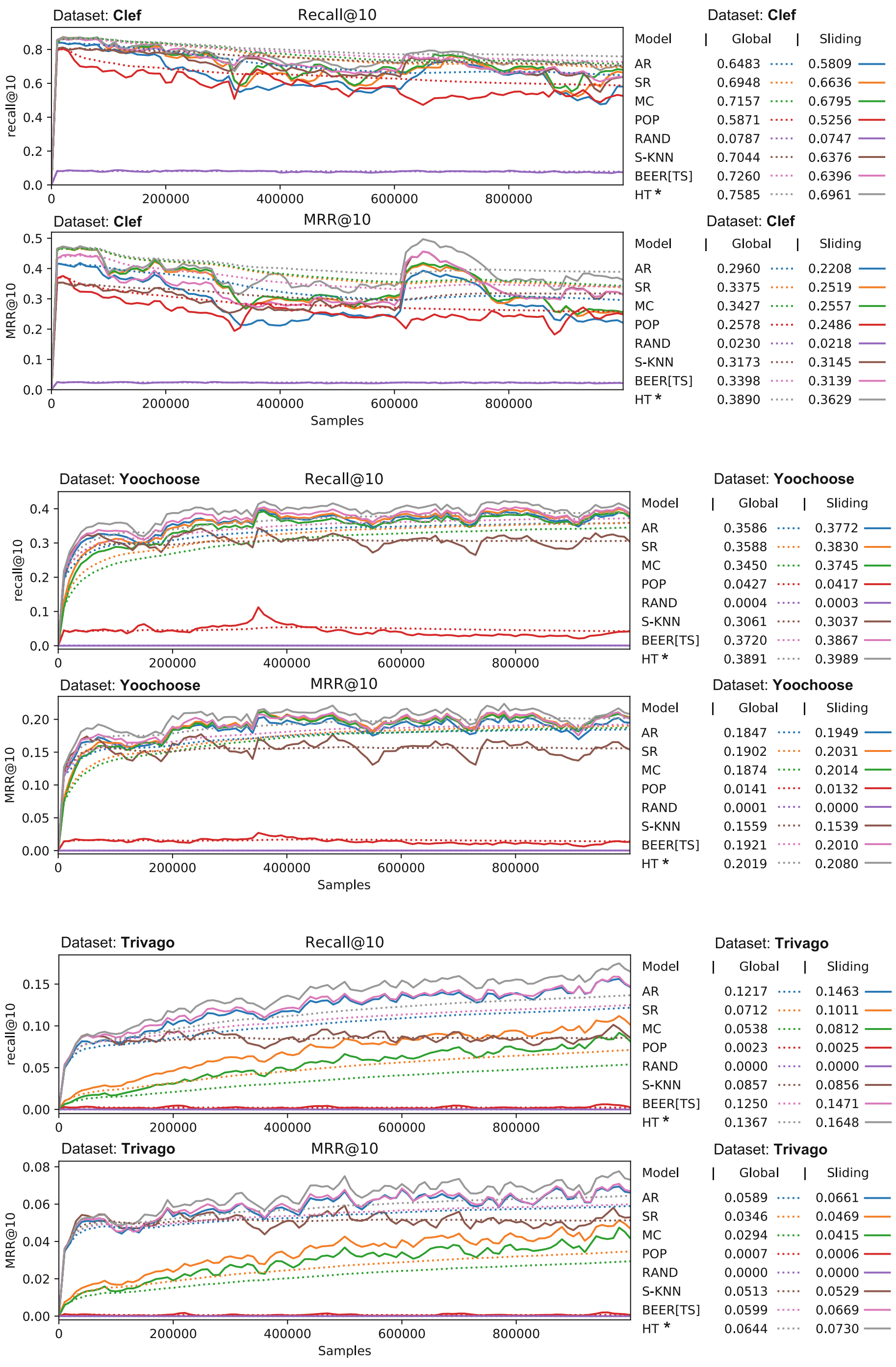
4.2 Prequential Evaluation Setup

We consider the task of online *next-item recommendation*, where the goal is to suggest the list of most probable items to appear in the next session event. We track Recall@10 and MRR@10 during the entire run of the simulation using the real-time visualizer provided by the framework, which reports the global and the current (sliding) averages. Time horizon for each simulation is set to 1M events, with evaluation and observation window sizes of 10K and 50K events, respectively. The latter size was chosen in consideration of sufficient (sliding) session history and a reasonable memory/runtime overhead. We use a separate validation set of 100K events (preceding those of the simulation) for hyperparameter tuning. The evaluation itself is performed from pure cold-start, with no model pre-training involved. Sessions of size 1 are excluded from the evaluation. For our simulations, we use Intel Core i7 CPU @ 2.80 GHz and 16 Gb RAM.

4.3 Model Setup

We evaluate the models presented in Sect. 3, with an addition of a Random classifier that sets the lower bound for performance. Below we briefly outline the optimal model configurations after the hyperparameter tuning step.

Rule-based (AR, MC, SR) models operate on a global scale, whereas the popularity-based one (POP) works within a sliding window. S-kNN uses Cosine similarity, and $k = 100$ (Trivago), $k = 200$ (Clef), $k = 300$ (Yoochoose). We also use *recent session sub-sampling* [8], with sub-sample sizes of 500 (Clef), 1000 (Trivago), and 1500 (Yoochoose). BEER[TS] includes AR, MC, SR, POP, and S-kNN as components of the ensemble. The HT wrapper uses weights $w_{MC} = 3$ (Yoochoose), $w_{MC} = 5$ (Clef, Trivago), and $w_{inv} = 0.01$ (Clef), $w_{inv} = 0.9$ (Yoochoose, Trivago). The HoeffdingTree class of `Scikit-Multiflow` is instantiated with leaf_prediction='nb' to enable Naive Bayes prediction at the leaves. The preconfigured experiments are provided in `FlowRec`'s code base for reproducibility.



Dataset: **Clef** (Recall@10)

Model	Global		Sliding	
AR	0.6483	····	0.5809	——
SR	0.6948	····	0.6636	——
MC	0.7157	····	0.6795	——
POP	0.5871	····	0.5256	——
RAND	0.0787	····	0.0747	——
S-KNN	0.7044	····	0.6376	——
BEER[TS]	0.7260	····	0.6396	——
HT *	0.7585	····	0.6961	——

Dataset: **Clef** (MRR@10)

Model	Global		Sliding	
AR	0.2960	····	0.2208	——
SR	0.3375	····	0.2519	——
MC	0.3427	····	0.2557	——
POP	0.2578	····	0.2486	——
RAND	0.0230	····	0.0218	——
S-KNN	0.3173	····	0.3145	——
BEER[TS]	0.3398	····	0.3139	——
HT *	0.3890	····	0.3629	——

Dataset: **Yoochoose** (Recall@10)

Model	Global		Sliding	
AR	0.3586	····	0.3772	——
SR	0.3588	····	0.3830	——
MC	0.3450	····	0.3745	——
POP	0.0427	····	0.0417	——
RAND	0.0004	····	0.0003	——
S-KNN	0.3061	····	0.3037	——
BEER[TS]	0.3720	····	0.3867	——
HT *	0.3891	····	0.3989	——

Dataset: **Yoochoose** (MRR@10)

Model	Global		Sliding	
AR	0.1847	····	0.1949	——
SR	0.1902	····	0.2031	——
MC	0.1874	····	0.2014	——
POP	0.0141	····	0.0132	——
RAND	0.0001	····	0.0000	——
S-KNN	0.1559	····	0.1539	——
BEER[TS]	0.1921	····	0.2010	——
HT *	0.2019	····	0.2080	——

Dataset: **Trivago** (Recall@10)

Model	Global		Sliding	
AR	0.1217	····	0.1463	——
SR	0.0712	····	0.1011	——
MC	0.0538	····	0.0812	——
POP	0.0023	····	0.0025	——
RAND	0.0000	····	0.0000	——
S-KNN	0.0857	····	0.0856	——
BEER[TS]	0.1250	····	0.1471	——
HT *	0.1367	····	0.1648	——

Dataset: **Trivago** (MRR@10)

Model	Global		Sliding	
AR	0.0589	····	0.0661	——
SR	0.0346	····	0.0469	——
MC	0.0294	····	0.0415	——
POP	0.0007	····	0.0006	——
RAND	0.0000	····	0.0000	——
S-KNN	0.0513	····	0.0529	——
BEER[TS]	0.0599	····	0.0669	——
HT *	0.0644	····	0.0730	——

Fig. 3. Performance charts for Clef (top), Yoochoose (middle), and Trivago (bottom)

4.4 Results

The results of the simulation for the three datasets are presented in Fig. 3. The top performing model in each case is marked with an asterisk. Note that the column ‘Sliding’ contains the averages of the *very last* evaluation window.

The HT wrapper consistently achieves higher recall and MRR than the other baselines on all three datasets. This proves the effectiveness of combining item-to-item sequential patterns, and utilizing the entire user session at prediction time in an ensemble-like manner. The HT wrapper also happens to be noticeably faster than its main rival, BEER[TS] (in its current configuration). The total running times for each model are recorded by the framework. As a point of reference, we consider the upper limit of 100 ms per recommendation prescribed by the CLEF NewsReel challenge [10]. Table 2 reports average response times per recommendation request for each model. We observe that all runtimes fall within the recommended limit. The real-time visualization offers the possibility to diagnose potential issues at an early stage. For instance, the performance charts for Yoochoose and Trivago make it obvious that the inclusion of POP to BEER[TS] is not justified, and hence it can be dropped from the ensemble to gain speed. The live monitoring of model evolution helps to see how algorithms behave relative to each other on various segments of the dataset, as well as to better understand the peculiarities of the dataset itself. For example, the above two charts clearly show the stagnation of S-kNN after leaving the initial cold-start segment (≈0–50K), while other models continue to learn. We also see that Clef exhibits a more dynamic domain (news) with more profound concept drift, which makes learning more challenging. It is the only dataset where the popularity recommender has decent performance, surpassing other models on certain data segments. The Clef chart also reveals the ‘easy’ portion of the dataset (≈600K–800K), where most algorithms (but not POP) boost their performance.

Table 2. Average recommendation time (msec) per model

Dataset	Model							
	AR	SR	MC	POP	RAND	S-kNN	BEER[TS]	HT
Clef	0.270	0.213	0.204	0.187	0.204	26.543	27.858	2.542
Yoochoose	0.744	0.623	0.476	3.365	0.313	2.434	13.331	9.629
Trivago	4.276	4.093	4.082	11.380	1.150	5.675	85.163	23.036

5 Conclusion

We introduce `FlowRec` — a new recommendation framework for streaming session data developed on top of `Scikit-Multiflow`. It serves as a testbed for streaming recommendation models by offering prequential evaluation with real-time performance monitoring. One advantage of prototyping in `FlowRec` is the ability to ‘wrap’ various stream learners provided by `Scikit-Multiflow`, thus treating them as black boxes. We demonstrate how to develop such a wrapper for the HoeffdingTree class, capable of generating accurate session-based recommendations on evolving data streams. The framework will be further extended with additional evaluation protocols, metrics, and algorithms.

References

1. Ben-Shimon, D., Tsikinovsky, A., Friedmann, M., Shapira, B., Rfokach, L., Hoerle, J.: Recsys challenge 2015 and the yoochoose dataset. In: Proceedings of the 9th ACM Conference on Recommender Systems (RecSys 2015), pp. 357–358. ACM (2015)
2. Brodén, B., Hammar, M., Nilsson, B.J., Paraschakis, D.: A bandit-based ensemble framework for exploration/exploitation of diverse recommendation components: an experimental study within e-commerce. ACM Trans. Interact. Intell. Syst. **10**(1), 4:1–4:32 (2019)
3. Diaz-Aviles, E., Drumond, L., Schmidt-Thieme, L., Nejdl, W.: Real-time top-n recommendation in social streams. In: Proceedings of the Sixth ACM Conference on Recommender Systems (RecSys 2012), pp. 59–66. ACM (2012)
4. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2000), pp. 71–80. ACM (2000)
5. Frigó, E., Pálovics, R., Kelen, D., Kocsis, L., Benczúr, A.: Alpenglow: open source recommender framework with time-aware learning and evaluation. In: 2017 Poster Track of the 11th ACM Conference on Recommender Systems (Poster-Recsys 2017), pp. 1–2. CEUR-WS.org (2017)
6. Guo, L., Yin, H., Wang, Q., Chen, T., Zhou, A., Quoc Viet Hung, N.: Streaming session-based recommendation. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2019), pp. 1569–1577. ACM (2019)
7. Hidasi, B., Karatzoglou, A., Baltrunas, L., Tikk, D.: Session-based recommendations with recurrent neural networks. CoRR abs/1511.06939 (2015)
8. Jannach, D., Ludewig, M.: When recurrent neural networks meet the neighborhood for session-based recommendation. In: Proceedings of the 11th ACM Conference on Recommender Systems (RecSys 2017), pp. 306–310. ACM (2017)
9. Jugovac, M., Jannach, D., Karimi, M.: Streamingrec: a framework for benchmarking stream-based news recommenders. In: Proceedings of the 12th ACM Conference on Recommender Systems (RecSys 2018), pp. 269–273. ACM (2018)
10. Kille, B., et al.: Stream-based recommendations: online and offline evaluation as a service. In: Mothe, J., et al. (eds.) CLEF 2015. LNCS, vol. 9283, pp. 497–517. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24027-5_48
11. Kitazawa, T.: Flurs: a python library for online item recommendation. https://takuti.me/note/flurs/ (2017). Accessed 03 November 2019
12. Knees, P., Deldjoo, Y., Moghaddam, F.B., Adamczak, J., Leyson, G.P., Monreal, P.: Recsys challenge 2019: session-based hotel recommendations. In: Proceedings of the 13th ACM Conference on Recommender Systems (RecSys 2019), pp. 570–571. ACM (2019)
13. Ludewig, M., Jannach, D.: Evaluation of session-based recommendation algorithms. User Model. User Adapt. Interact. **28**(4–5), 331–390 (2018). https://doi.org/10.1007/s11257-018-9209-6

14. Montiel, J., Read, J., Bifet, A., Abdessalem, T.: Scikit-multiflow: a multi-output streaming framework. J. Mach. Learn. Res. **19**(72), 1–5 (2018)
15. Quadrana, M.: Algorithms for sequence-aware recommender systems. Ph.D. Thesis, Politecnico di Milano (2017)
16. Scriminaci, M., et al.: Idomaar: a framework for multi-dimensional benchmarking of recommender algorithms. In: Guy, I., Sharma, A. (eds.) RecSys Posters. CEUR Workshop Proceedings, CEUR-WS.org (2016)
17. Srimani, P., Patil, M.M.: Performance analysis of hoeffding trees in data streams by using massive online analysis framework. Int. J. Data Min. Model. Manage. **7**(4), 293–313 (2015)
18. Subbian, K., Aggarwal, C., Hegde, K.: Recommendations for streaming data. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (CIKM 2016), pp. 2185–2190. ACM (2016)
19. Tan, Y.K., Xu, X., Liu, Y.: Improved recurrent neural networks for session-based recommendations. In: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS 2016), pp. 17–22. ACM (2016)
20. Vinagre, J., Jorge, A.M., Gama, J.: Evaluation of recommender systems in streaming environments. In: ACM RecSys Workshop on Recommender Systems Evaluation: Dimensions and Design (REDD 2014), pp. 393–394. ACM (2014)