Recommending News Articles in the CLEF News Recommendation Evaluation Lab with the Data Stream Management System Odysseus

Cornelius A. Ludmann
cornelius.ludmann@uni-oldenburg.de

University of Oldenburg
Department of Computer Science
Escherweg 2, 26121 Oldenburg, Germany

Abstract. A crucial aspect of recommending news articles is the relevance of currentness of articles. Every day, news portals add plenty of new articles. Typically, users are more interested in recently published articles (or articles that provide background information to recently published articles) than in older ones. That leads to the demand to continuously adapt the set of recommendable items in a recommender system.
In this paper, we share our experiences with the usage of the generic and open source Data Stream Management System (DSMS) *Odysseus* as a Recommender System in the CLEF NewsREEL 2017. We continuously calculate the *currently* most read articles based on a stream of impression events. Our approach uses operators of a stream-based variant of the relational algebra that respects validity intervals of events. This allows us to continuously calculate the K most popular articles (*Top-K set* regarding to the number of views) in a sliding time window based on well established relational operations. The flexible composition of operators allows us to variate, e.g., the grouping of impressions to get different recommendation sets for different user groups or the exclusion of articles the user already knows.

Keywords: Recommender Systems, Data Stream Processing

1 Introduction

A recommender system as a component of modern information systems provides a user with a set of items (e.g., news articles, movies, or products) which might be of interest to him or her. The challenge is to select a small sample from a big collection of items that hopefully meets the taste of the user. A common approach is to learn from previous actions (e.g., page impressions of a web news portal) to estimate the usefulness of items. The K most useful items are presented to the user as recommendations.

The CLEF News Recommendation Evaluation Lab [4] (NewsREEL) allows researchers to evaluate recommender systems with real user data in real-time. They provide a platform, called *Open Recommendation Platform* (ORP), that

sends the participants events (impressions and clicks on recommendations) of real commercial news portals. The participants have to analyze the data and must answer requests for recommendations with a set of news articles within a timespan of 100 ms. The recommendations are displayed on the news portal next to the article a user currently reads. For each participant, the organizers measure the clicks on the recommendations and calculates the *Click Through Rate* (CTR).

In order to participate, a participant needs to implement a server that processes the events and replies to recommendation requests. The organizers provide a framework[1] that parses the events and lets one implement his/her own algorithms. However, using that framework means you need to take care of the data management and processing of potentially unbounded data streams of events.

In this paper we present our solution that uses the general purpose *Data Stream Management System* (DSMS) called *Odysseus*[2] [1]. It adapts the concepts of Database Management Systems (DBMS) to process data streams with continuous queries.

2 Background

In this section we give a summary of the ORP platform, the CLEF NewsREEL, and the DSMS Odysseus.

2.1 ORP and CLEF NewsREEL

ORP as the technical platform for NewsREEL provides events of user activity of different news web portals in real-time. Participants are invited to use these events to calculate recommendations. The recommendations have to be returned upon request within 100 ms. ORP selects a set of recommendations at random out of all valid recommendation responses of all participants and displays the recommendations to the users next to the articles. As evaluation criteria, ORP uses *click through rate* (CTR). It counts how many of the requested recommendations get clicks by the users of the news portals. The fraction of clicks divided by the number of displayed recommendation sets is the CTR.

Participants have to subscribe the following data streams:

1. Notifications about user events (e.g., a user reads a news article).
2. Requests for recommendations, to which the participants have to respond to.
3. Notifications about new or updated news articles.
4. Error notifications (e.g., a participant response could not be parsed).

In our approach we use the first two streams: The user events stream provides impressions for news articles in real-time. It comprises the news article ID, publisher ID, user/session ID, user geolocation code, and other information about the impression. That is our source of data to calculate recommendation sets.

[1] https://github.com/plista/orp-sdk-java
[2] http://odysseus.uni-oldenburg.de/

The requests stream triggers the reply of an ordered set of article IDs—the recommendations. A request consists of similar attributes as the impressions: The article the user reads currently, the publisher of the article, the user, etc.

The events are pushed via HTTP POST request as JSON documents. The response of a request for recommendations has to be the article IDs intended to recommend to the user. For each participant ORP sends the data with an individual data rate. Participants that are able to process more data in time get a higher data rate. ORP determines the individual data rate by increasing it until the participant is not able to answer in time anymore. According to the organizers, each algorithm has to give at least 75 % of the amount of recommendations the baseline approach gave in order to be considered. That should ensure that the algorithms are able to handle the workload and allows the comparison of different approaches.

Table 1 shows the results of NewsREEL 2017 [6] as provided by the organizers. It shows the name of the recommender approach, the number of recommendations given, the number of clicks on recommendations, and the click through rate. Additionally, we added the last column that indicates if the approach gave at least 75 % of the baseline recommender (BL2Beat). Each recommender that did not meet this criteria has been grayed out. That applies to each recommender that gave less than 46,539 recommendations. Our approaches have been highlighted in bold face.

	Recommender	**# Recomm.**	**# Clicks**	**CTR**	**75 % BL**
	Riadi_NV_01	443	12	0.0271	no
	ORLY_KS	42,786	896	0.0209	no
1.	**ody4**	72,601	1,139	0.0157	yes
	IRS5	3,708	58	0.0156	no
2.	**ody5**	81,245	1,268	0.0156	yes
3.	**ody3**	59,227	813	0.0137	yes
4.	**ody2**	63,950	875	0.0137	yes
5.	IT5	68,582	925	0.0135	yes
6.	eins	61,524	817	0.0133	yes
7.	yl-2	60,814	747	0.0123	yes
8.	WIRG	49,830	600	0.0120	yes
9.	**ody1**	68,768	810	0.0118	yes
10.	BL2Beat	62,052	726	0.0117	yes
11.	RIADI_pn	77,723	879	0.0113	yes
12.	IL	79,120	813	0,0103	yes
13.	RIADI_nehyb	75,535	764	0.0101	yes
	Has logs	816	6	0.0074	no
	ody0	23,023	166	0.0072	no
	RIADI_hyb	349	2	0.0057	no

Table 1: Results of CLEF NewsREEL 2017

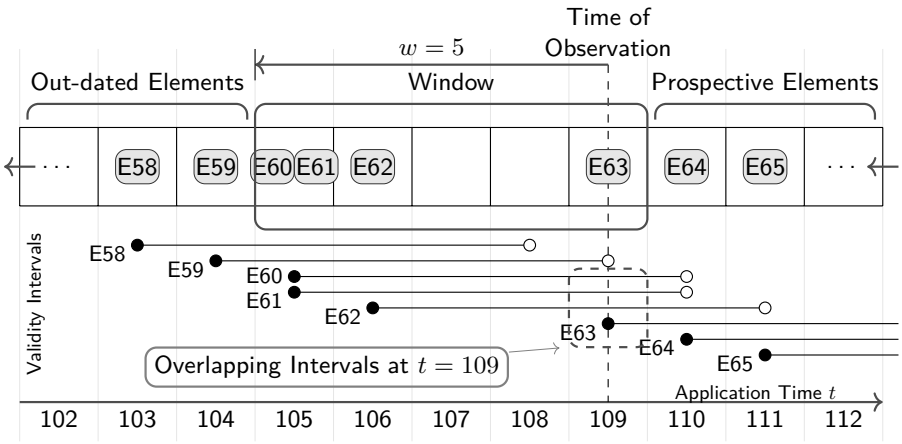


Fig. 1: Snapshot Reducibility

2.2 DSMS—The Odysseus System

The concept of a DSMS bases on the relation algebra of Database Management Systems (DBMS). Instead of managing a static database of relations, a DSMS manages volatile data that is pushed to the system by active sources. That results in a potentially unbounded sequence of elements—the *data stream*. Similar to a DBMS, a user of a DSMS writes a query by the use of a query language and the system builds a *Query Execution Plan* (QEP), which consists of reusable operators. Each operator is responsible for a certain operation. Examples of common operators are SELECTION (filtering of elements), PROJECTION (filtering of attributes of elements), JOIN (combining of elements of two data streams), and AGGREGATION (combining a set of elements to a single resulting value).

A relational DSMS extends the concept of a relational DBMS by time-aware and incremental data processing. The underlying concept has been extensively studied, e.g. in [3, 2, 5, 1].

We treat events as tuples with a fixed schema. Each event occurs at a certain point in time $t \in T$ with T as set of discrete and ordered time instances, e.g., the UNIX timestamp or an arbitrary application/event time (cf. [5]).

The most important differences to a DBMS are that a DSMS holds its data in memory and supports (sliding time) windows.[3] That allows to define a finite set of valid tuples for each point in time $t \in T$, e.g., to calculate a moving average of all elements not older than 60 seconds. The term *snapshot reducibility* [5] claims that the resulting tuples of a query of a stream processing system at a specific

[3] In this paper we limit ourselves to sliding time windows. However, there are also other types of windows, e.g., element windows, tumbling windows etc. that are supported by our DSMS.

```
/* Input and Output Stream Definitions */
CREATE STREAM events (type STRING, articleid INT,
  publisherid INT, userid INT, userloc INT, ...) ...;
CREATE STREAM requests (publisherid INT, userid INT,
  userloc INT, ...) ...;
CREATE SINK recommendations (recs LIST_INT) ...;

/* Continuous Query Definition */
CREATE VIEW impressions FROM (
  SELECT articleid, publisherid
  FROM events [SIZE 30 Minutes TIME]
  WHERE type = "impression" AND articleid > 0
);
CREATE VIEW counted_impressions FROM (
  SELECT publisherid, articleid, count(*) AS counts
  FROM impressions GROUP BY publisherid, articleid
);
CREATE VIEW topk_sets FROM (
  SELECT publisherid,
         nest(articleid) AS most_popular_articles
  FROM counted_impressions
  GROUP BY publisherid ORDER BY counts DESC
  GROUP LIMIT 6
);

/* Join of Requests and TopK Sets */
STREAM TO recommendations FROM (
  SELECT topk_sets.most_popular_articles AS recs
  FROM topk_sets, requests [SIZE 1 TIME] AS req
  WHERE topk_sets.publisherid = req.publisherid
);
```

Listing 1.1: CQL Query Definition

point in time t are the same as the resulting tuples of a corresponding query of a DBMS over the set of all valid tuples at t.

In this paper we use the concept of validity intervals (cf. [5]). This approach adds an additional operator to the relational algebra—the WINDOW operator $\omega_w(R)$. It assigns validity intervals to elements of a relational stream R. E.g., a sliding time window of size w states that the processing at a point in time t' should respect all events not older than $t' - w$. Therefore, the operator sets a (half open) validity interval $[t_s, t_e)$ with $t_s = t$ and $t_e = t + w$ to an event that has been arisen at time t. Thus, the other operators do not need to know the window size: The calculation of a result at time t' need to respect all events where $t' \in [t_s, t_e)$ for the certain event. These are all events that are not older than $t' - w$.

Fig. 1 illustrates the concept of snapshot reducibility with validity intervals. It shows the validity intervals for the elements and a sliding time window of size $w = 5$. It depicts a part of a data stream from $t = 102$ to $t = 112$ with the elements E58 to E65. A sliding time window of size $w = 5$ defines that at each point in time t all elements that are not older than $t - 5$ should be considered as valid. That means at point $t = 109$ in Fig. 1 all elements not older than $t' = 105$ are valid. That corresponds to a window from 105 to 109.

To process the events of a data stream, the user writes one or more queries. In contrast to queries of a DBMS, a query will be executed until the user stops it explicitly. Such a query is called a *continuous query* since it processes the data continuously. To write a query, the users take advantage of a query language, as the SQL-like stream-based query language CQL [2] or the functional query language PQL [1].

Similar to a DBMS, the stream processing system translates the query to a *Query Execution Plan* (QEP; in the streaming context also called *data flow graph*). A QEP is a directed graph of operators. It determines the order of processing and can be optimized, e.g., by changing the order of operators.

In addition to the actual stream processing, a DSMS is also responsible for resource and user management, authentication, authorization, accounting, etc.

3 General Approach

To illustrate how to use a DBMS as a RecSys in the CLEF NewsREEL challenge, we present an approach that recommends the most popular articles of the last 30 minutes as basis query for further improvements. Listing 1.1 shows the definition of the query in CQL. This is very similar to SQL and consists of a data definition (DDL) and a data manipulation (DML) part.

The first part is the definition of the input streams (Lines 2-5) and the output stream (called *sink*, Line 6). This is similar to a `CREATE TABLE` statement of a DBMS. It consists of the name of the stream (e.g., `events`) and a schema definition (truncated in Listing 1.1). The stream or sink definition includes also information about the data connection (e.g., HTTP, TCP) and data format (e.g., CSV, JSON). This is omitted in Listing 1.1 because this is out of the scope of this paper. After parsing the incoming data each event results in a tuple with a fixed schema.

The second part is the definition of the actual queries. It consists of three parts: the pre-processing of the events (lines 9-13), the counting of the number of impressions for each article (lines 14-17) and the aggregation of the six most popular articles for each publisher (lines 18-24). Each of this is expressed as a view definition, which allows to reference the results in the subsequent queries. The `impressions` view selects the article ID and publisher ID of all events of type "impression" that have an article ID. Additionally, it defines a sliding time window of a fixed size (e.g., 30 minutes). The result is used in the `counted_impressions` view. It counts the number of impressions (in the time window) of each unique pair of publisher and article ID. When a new impression arrives or an event gets invalid, the count changes and a new tuple is produced that updates the previous value. The `topk_sets` view aggregates the counted impressions to a list of articles IDs of the six most popular articles for each publisher.

The third part uses the list of most popular articles to answer the recommendation requests (lines 27-31). For this, we join a request event with an event of the `topk_sets` view for the events where publisher ID of request and recommendation set are the same. The stream-based join operator solely joins events

that are valid at the same time (which means they have overlapping validity intervals). Because the aggregation of the `topk_sets` view updates the most popular articles set when they change, all resulting tuples of the same publisher have non-overlapping validity intervals: For each point in time there is exactly one valid aggregation result for the same group. By defining a sliding time window of size 1 (validity of 1 time slice) over the stream of requests, the join assigns exactly one set of article IDs to each request event.

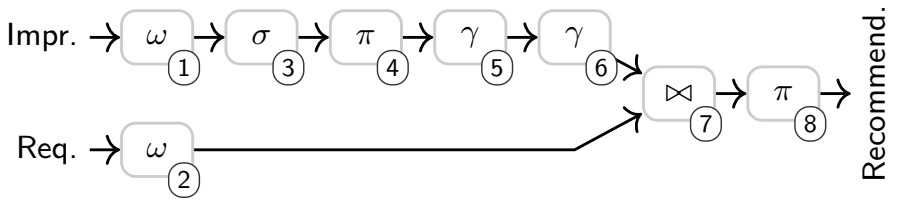


Fig. 2: Query Execution Plan

The resulting QEP is depicted in Fig. 2. It consists of the following types of operators:

- A WINDOW $\omega_w(R)$ as described in the previous section.

  We use this operator to control how long the impression events are considered by the subsequent aggregation operator (①) and to limit the validity of a recommendation request to 1 time slice (②) in order to join it with exactly one set of recommendations.
- A SELECTION $\sigma_\varphi(R)$ removes all tuples $t \in R$ for which the propositional expression φ does not hold.

  We use the SELECTION to filter the impressions from the event stream (③).
- A PROJECTION $\pi_{a_1,\ldots,a_n}(R)$ restricts the set of attributes of each tuple $t \in R$ to the attributes $\{a_1, \ldots, a_n\}$. An extended version, called MAP, allows to use functions (e.g., $\pi_{f(a_1),a_2,f(a_3,a_4),\ldots,a_n}(R)$) that are invoked with one ore more attributes.

  The PROJECTION is used in our example to filter the needed attributes of the impression events and of the recommendation responses (④ and ⑧).
- An AGGREGATION $\gamma_{G,F}(R)$ takes a set of tuples $t \in R$ and returns for each group defined by the grouping attribute $g \in G$ a single tuple as result that contains the results of the aggregate functions $f \in F$. Typical aggregate functions are SUM, COUNT, AVG, MAX, and MIN. The stream-based variant continuously aggregates tuples that lie in the same window (which means they have overlapping validity intervals).

  We use the AGGREGATION to count the number of impressions of each article (⑤) and to nest the 6 most viewed articles into one outgoing tuple (⑥).

```
State[] state   % the current state for each group
Event[] events  % events in the current window

for each incoming Event e:
  for each Event r in events with r.endts ≤ e.startts:
    state[group(r)] ← remove(state[group(r)], r)
    events ← events − r
    output: eval(state[group(r)])
  end for
  state[group(e)] ← add(state[group(e)], e)
  output: eval(state[group(e)])
  events ← events + e
end for
```

Listing 1.2: Algorithm of the aggregation operator.

– A JOIN $R \bowtie_\theta S$ combines the tuples of R and S for which a condition θ holds. The stream-based variant claims also that the tuples have overlapping time intervals. The validity interval of a resulting tuple is the intersection of the validity intervals of the input tuples.

  The JOIN (⑦) is responsible for the matching of the requests to the recommendation sets.

The important parts of the query are the calculating of a ranking score (AGGREGATION ⑤), the building of the recommendation sets (AGGREGATION ⑥), and the matching of requests and recommendation sets (JOIN ⑦). These operators are heavily influenced by validity intervals of the events, that are assigned by the window operator.

The calculation of the recommendation sets is implemented by two AGGREGATION operators: One to calculate the popularity of an article and one to combine the most popular articles to a recommendation set. Our AGGREGATION operator is data driven. That means it outputs an updated result for every incoming event. The operator internally holds a state s for each group and updates this state using an aggregation function $\text{add}(s, e) \mapsto s$ for each incoming event e. For the COUNT function in Listing 1.1 the aggregation function is defined as: $\text{add}(s, e) = s + 1$.

Since the aggregation is defined over a sliding time window the operator has to remove all values that do not lie in the window anymore. For that, all events whose end timestamp of their validity intervals are lower or equal than the start timestamp of the incoming event have to be removed. This is done before adding a new event by using a function $\text{remove}(s, e) \mapsto s$, e.g., for COUNT: $\text{remove}(s, e) = s - 1$.

After each change of the state the operator outputs a result by calling the function $\text{eval(s)} \mapsto \text{e}$, which calculates the resulting event based on the state. For COUNT, it just outputs the state value s. Other functions as for example AVG need to calculate the resulting value. Consider a state for AVG that consists of a count and a sum, the eval function is $\text{eval}(s) = \frac{s.sum}{s.count}$. Listing 1.2 illustrates the algorithm of the aggregation operator.

The second AGGREGATION operator uses the NEST function with $\text{add}(s, e) = s.\,\text{insertSorted}(e)$ (sorted by `count`), $\text{remove}(s, e) = s.\,\text{remove}(e)$, and $\text{eval}(s) =$

$s.\text{sublist}(0, 6)$. The output is an ordered set of the article IDs of the 6 events on the top.

Since the recommendation sets are calculated incrementally and continuously, the calculation of the response to a request is the executing of the JOIN operator. This leads to very short latencies. The join holds for each group (here the publisher ID) the latest known recommendation set in a hash map. The operator appends the matching recommendation sets to each request and gives the result to the sink that transfers it back to the inquirer.

4 Variations

During the CLEF NewsREEL 2017, we evaluated six different approaches that are based on the general approach presented in the previous section. Table 2 shows an overview over our approaches and their resulting CTRs.

The approach *ody0* uses the aggregation function NEST over a sliding time window of 60 min. to aggregate a set of unique item IDs that have been read in the past 60 min. After that, a MAP function is used to randomly draw 6 items to recommend them. This approach acts as our internal baseline.

All other approaches follow the same pattern. Given a data stream of article IDs and optional further attributes:

1. Define a sliding time window of size w over the data stream. (In our DSMS, this is done by annotating validity intervals.)
2. Partition the data by the article IDs and optional other attributes $p_1, p_2, \ldots, p_n$.
3. Apply an aggregation function over each partition that calculates a score for each item in each partition (over the sliding time window). (In our approach we used the count function. In case of explicit item rating, the average function would be an alternative.)
4. Optional: Combine the results of different approaches to prevent underfull recommendation sets.

This results in a incrementally and data-driven calculated stream of recommendation sets for each partition.

The approaches *ody1* and *ody2* partition the data by publisher IDs as described in the previous section. They differ solely in the window size. *ody1* considers all impressions of the past 30 min., *ody2* of the past 5 min. The evaluation shows, that the 5 min. window leads to a higher CTR than the 30 min. window.

ody3 uses the same window size as *ody2* but adds as partition attribute the user location. Because more partitions lead to less data in each partition, our tests show that a larger window size than 5 min. is necessary to have enough data in each partition and each window. This approach reaches a similar CTR than *ody2*.

In contrast, *ody4* and *ody5* calculate the recommendations by aggregating the previously successfully given recommendations of the past 12 hours. This information is part of the user events stream. For each news article i we calculated the set of news articles that have been successfully recommended to

visitors of item i the most. Additionally, *ody5* fills the recommendation set up with recommendations of *ody1* in case that there are less than six successfully recommended items in the past 12 hours.

	Approach	CTR
ody0:	Random sample of 60 min. sliding time window.	0.0072
ody1:	Most viewed articles of 30 min. sliding time window partitioned by publisher.	0.0118
ody2:	Most viewed articles of 5 min. sliding time window partitioned by publisher.	0.0137
ody3:	Most viewed articles of 30 min. sliding time window partitioned by publisher and user location.	0.0137
ody4:	Most clicked recommendations of 12 hour sliding time window partitioned by item.	0.0157
ody5:	Most clicked recommendations of 12 hour sliding time window partitioned by item filled up with most viewed articles of 30 min. sliding time window partitioned by publisher.	0.0156

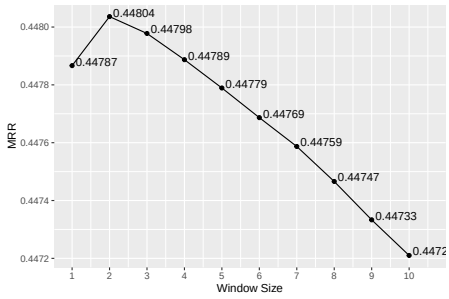
Table 2: Overview over our approaches in the CLEF NewsREEL 2017

5 Evaluation of the Window Size

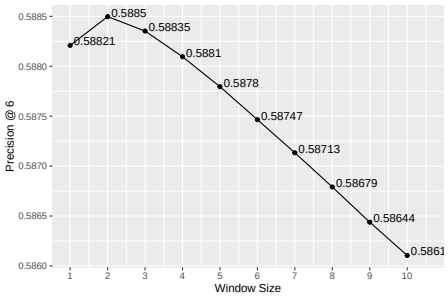
A crucial parameter is the window size of the impression events. Since we want to count the *current* views of each article we have to figure out which time span leads to a set of impression events that represent the *current* popularity of articles: Is the size too large the system is not sensitive to popularity changes (concept drifts). Is it too small there is not enough data to distinguish the popularity of articles.

To determine an appropriate window size we conducted an experiment. We ran 21 queries as shown in Listing 1.1 with different window sizes in parallel over the same stream (1 to 10 min, 20 min, 30 min, 40 min, 50 min, 60 min, 90 min, 2 hrs, 3 hrs, 6 hrs, 12 hrs, 24 hrs). Similar to the *Interleaved Test-Then-Train* (ITTT) evaluation method we calculated the reciprocal rank of each article of an impression event before it has influenced the recommendation set. That gave us a *mean reciprocal rank* (MRR) for each query resp. window size. A higher MRR means there are frequently viewed articles more often near the top. Additionally, we calculated how often an article is part of the top 6 of the most popular items in the different time windows (Precision@6).

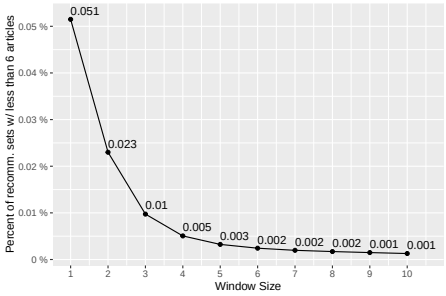
Because the data rates of the publishers are different (some publishers have many more impressions than others) we evaluated the window sizes for each publisher separately.



(a) Mean Reciprocal Rank



(b) Precision @ 6



(c) Percent of Underfull Recomm. Sets

Fig. 3: Mean Reciprocal Rank, Precision@6, and underfull recommendation sets for window sizes 1 - 10 min and publisher with ID 35774.

Fig. 3 shows the result for the publisher with ID 35774 (window sizes 1 to 10 min). As you can see in Fig. 3a, a window of 2 min. leads to the highest MRR of 0.44804. Correspondingly, the Precision@6 in Fig. 3b shows that about 59% of each impression the user views an article that is part of the 6 most popular articles of the last 2 min. For this publisher, lower or higher window sizes lead to lower values.

Besides the metrics MRR and Precision@6, we measured how often a window does not have impressions of at least 6 distinct articles so that the recommendation sets have less than 6 recommendations (underfull recommendation sets). Fig. 3c shows the fraction of underfull recommendation sets. As expected, the fraction of underfull recommendation sets decreases with higher window sizes. Overall, the fraction of underfull recommendation sets for publisher 35774 is pretty low.

Even a window size of 2 min. has just 0.023% recommendation sets with less than 6 recommendations.

The results of the other publisher follow this pattern, even though publisher with lower data rates get better results with bigger window sizes. As a tradeoff between different publishers we made good experiences with a window size of five minutes.

6 Conclusions

In this paper we presented an approach for the CLEF NewsREEL 2017. We implemented a RecSys by using a generic Data Stream Management System. By analyzing impression events of users we calculated a set of recommendations based on the popularity in a given time window. To find an appropriate time window we evaluated different sizes in parallel.

References

1. Appelrath, H.J., Geesen, D., Grawunder, M., Michelsen, T., Nicklas, D.: Odysseus: A highly customizable framework for creating efficient event stream management systems. In: DEBS'12. pp. 367–368. ACM (2012)
2. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: semantic foundations and query execution. VLDB Journal 15(2), 121–142 (2006)
3. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: PODS 2002. pp. 1–16. ACM (2002)
4. Hopfgartner, F., Brodt, T., Seiler, J., Kille, B., Lommatzsch, A., Larson, M., Turrin, R., Serény, A.: Benchmarking news recommendations: The clef newsreel use case. In: ACM SIGIR Forum. vol. 49, pp. 129–136. ACM (2016)
5. Krämer, J., Seeger, B.: Semantics and implementation of continuous sliding window queries over data streams. ACM TODS'09 34(1), 4 (2009)
6. Lommatzsch, A., Kille, B., Hopfgartner, F., Larson, M., Brodt, T., Seiler, J., Özgöbek, Ö.: Clef 2017 newsreel overview: A stream-based recommender task for evaluation and education. In: CLEF 2017. Springer Verlag (2017)