Online Recommender Systems based on Data Stream Management Systems

Cornelius A. Ludmann
University of Oldenburg
Department of Computer Science, Information Systems Group
Escherweg 2, 26121 Oldenburg, Germany
cornelius.ludmann@uni-oldenburg.de

ABSTRACT

In this paper, I present a novel approach for implementing a stream-based Recommender System (RecSys). I propose to add RecSys operators to an application-independent Data Stream Management System (DSMS) to allow writing continuous queries over data streams that calculate personalized sets of recommendations. That empowers RecSys providers to create a custom RecSys by writing queries in a declarative query language. This approach ensures a flexible and extendable usage of RecSys functions in different settings and benefits from matured features of DSMSs.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*information filtering, relevance feedback*

Keywords

recommender system; collaborative filtering; data stream management system; stream processing

1. INTRODUCTION

In real-world Recommender Systems (RecSys), ratings are produced continuously and in a temporal order by a large number of users. Recent research on RecSys has emerged model-based collaborative filtering algorithms for streaming data (so-called online or incremental collaborative filtering; e. g., [11]). These algorithms allow to add new ratings to existing models. Thus, the model can be updated more often and is able to express the recent interests of the users. Combined with methods of time-aware RecSys [6], these approaches allow the calculation of recommendations influenced by the *current* interests of the users instead solely based on general preferences.

There are some libraries that provide implementations of learning algorithms for RecSys models. However, it takes more than learning algorithms to operate a stream-based and time-aware RecSys. Such a system needs to manage and provide input and output data stream connections, parse and transform streaming data, organize the (transient) storage of the data, optimize and distribute the data processing, should allow a configuration of the RecSys, etc.

In this paper, I propose to add RecSys functions to a application-independent Data Stream Management System (DSMS). A DSMS is designed to continuously process data streams by the use of query plans. A query plan consists of operators that implement base functions (e. g., a stream-based variant of the relational algebra) as well as special tasks (e. g., data mining operators). Using a DSMS for a stream-based RecSys has the following advantages:

- The RecSys operators can be used by DSMS users in arbitrary queries written in a declarative query language. They can build a customized RecSys without the need of writing code.
- A lot of operations needed for a RecSys can be covered by existing operators. This includes the access to data sources and sinks, the calculation and aggregation of values like the model error for the RecSys evaluation, the selection of the top-K recommendations, etc.
- A RecSys can be enhanced by adding additional operators to the query. This can be useful to combine different data sources, to pre- and post-process data (e. g., normalization of input data), to include and process additional data (e. g., context data or linked open data), to combine different model learners for ensemble learning, etc.
- The RecSys developer can benefit from many DSMS features like query plan optimization, query sharing, parallelization and distribution of queries, scheduling, etc.

This leads us to the following research questions:

- How can a collaborative and model-based RecSys be implemented by a DSMS? How can a RecSys be split into DSMS operators to achieve a flexible composition of a RecSys by a declarative query language?
- How can a RecSys benefit from DSMS features? What are the limitations of this approach?
- Is it possible to efficiently implement different state-of-the-art RecSys methods to a DSMS? How can different implementations be compared?

The remainder of this paper is structured as follows: First, I introduce the background of my work (Sec. 2). Then, in Sec. 3, I present the current state of my approach. This includes a sample query plan for a RecSys, the introduction

RecSys'15, September 16–20, 2015, Vienna, Austria.

ACM 978-1-4503-3692-5/15/09 ...$15.00.
DOI: http://dx.doi.org/10.1145/2792838.2796544 .

of new operators, and a basic implementation that shows the feasibility. In Sec. 4, I give an overview of related work. Finally, I discuss plans for feature work (Sec. 5).

2. BACKGROUND

Rating Data and Recommendations

Let $U = \{u_1, \ldots, u_n\}$ and $I = \{i_1, \ldots, i_m\}$ be the sets of all users and items, respectively. Furthermore, let T be a discrete time domain with a total order, e. g., the set of all non-negative integers $T = \{0, 1, 2, 3, \ldots\}$ interpretable as UNIX timestamp. Every user u has interacted with a set of items $I_u \subset I$. Each interaction results in a rating score r that quantifies the interest of the user to the item he/she interacted with at a certain time $t \in T$. The rating might be explicitly given by the user or implicitly derived from the interaction. This leads to a set of tuples of known or observed ratings $\mathcal{R} := \{(u, i, r, t) \mid (u, i, r, t) \in U \times I \times \mathbb{R} \times T\}$ that we call the *rating data*. The challenge of a time-aware RecSys is to predict a rating $\hat{r}$ of an item i for a certain user u at a point in time t. In order to give recommendations for a user u, the RecSys predicts ratings for each item of a set of recommendation candidates for user u and recommends these with a minimum predicted ratings score and/or the K items with the highest ratings (top-K set). The set of recommendation candidates for user u includes usually all items not rated by u.

Data Streams and DSMSs

A data stream in our DSMS is a potentially infinite sequence of elements, each annotated with an half-open validity time interval $[t_s, t_e)$. The elements of every stream are ordered by t_s, so that t_s is monotonically, but not necessarily strictly, increasing. An operation that uses different elements of one or more data streams should process only these elements together that have overlapping validity intervals (elements that are valid at the same time). Due to the required order of stream elements, an operation can drop a stream element e when each involved stream has delivered an element with a t_s greater than or equal to t_e of e. In this case, no data stream will deliver another element whose validity interval overlaps with e. By default, each element becomes valid at the time t of creation (e. g., when a user rates an item) and will be valid forever, which results in an interval $[t, \infty)$.

DSMSs are developed to continuously process data streams. Similar to Database Management Systems (DBMSs), a user writes queries by the use of a declarative query language. The system parses the query and transforms it into a logical query plan of operators that determines the logical sequence of operations. This plan can be optimized (e. g., by changing the operator order) and is translated into a physical query execution plan, which consists of operators with an appropriate implementation. While traditional DBMSs focus on giving a precise answer by a one-time query on a fully known persistent data set, DSMSs typically use continuous queries to process streams of transient data in order to continuously give (often approximated) answers [4]. In contrast to a DBMS, a DSMS has no random access to the data. It accesses the elements sequentially in order of the occurrence in the stream. An element cannot be requested again unless it is explicitly held in memory (*one-pass paradigm*). Because a stream is potentially infinite, the number of stored elements needs to be limited to prevent a memory overflow [4].

3. CURRENT STATE

In this section, I introduce an approach for building a RecSys based on a DSMS. The RecSys has two input streams: one that provides the rating data as user-item-rating triples and one that provides requests for recommendations. An output stream delivers sets of recommendations for each request. Additionally, I define a data stream of model errors as an output of a continuous evaluation.

The stream elements are processed by operators of a query plan. To simplify writing customized RecSys queries, I introduce a set of logical RecSys operators that abstract from the actual physical implementation. During the query plan transformation, these operators were replaced by basis operators or translated to RecSys-specific physical operators. Where possible, existing basis operators were reused. The transformation of the logical query plan to physical operators can be customized by transformation rules (e. g., choosing the best implementation depending on the input data) or controlled by query parameters given by the DSMS user.

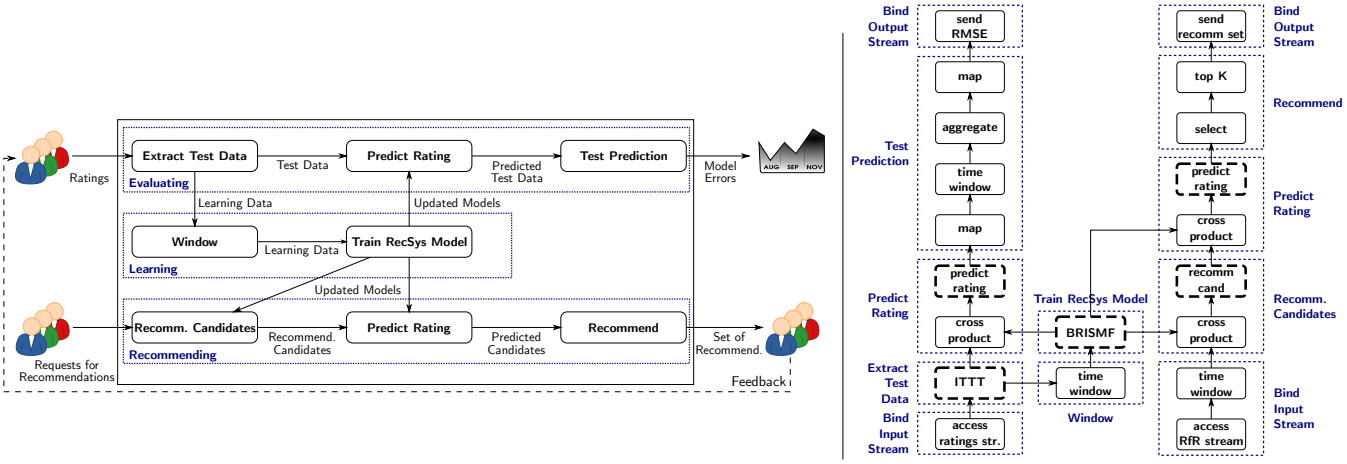
Logical Operators and Logical Query Plan

A minimal RecSys query uses the data of the ratings input stream to train a RecSys model and outputs recommendation sets for each request for recommendations by the use of the trained model. Additionally, it can be extended by operators that evaluate the recommendation quality. Figure 1a shows a logical query plan of a RecSys that can be used as a reference for custom queries. On the left we see the incoming and on the right the outgoing data streams as described above. The logical operators and the intermediate data flow (solid lines with arrows) are depicted in between. The RecSys operators are distinguished between learning, recommending, and evaluating operators.

The rating data stream is split by EXTRACT TEST DATA into two data streams: a stream with test data and a stream with learning data. The physical implementation of this operator is responsible for the evaluation methodology (e. g., *hold out* or *interleaved test-then-train* (ITTT); cf. [9]).

The learning data is used by TRAIN RECSYS MODEL to train a model. With every new learning tuple, this operator updates the model and outputs a copy for the subsequent operators. This operator must set the validity intervals of the outgoing models according to the following rules: (1) At every point in time, there is exactly one valid model. That means, the previous model needs to become invalid when a new model becomes valid. (2) All learning tuples that are used to train the model need to be valid in the complete validity interval of the model and there must exist no learning tuple that was not used for model training that is valid in the validity interval of the model. That means, the operator must output an updated model every time a learning tuple becomes valid or invalid. A preceding WINDOW operator allows to limit the validity of the learning data by modifying t_e. This can be useful to learn a model that focuses only on the newest ratings and to remove outdated ratings (to handle concept drifts). Additionally, it limits the ratings held in memory.

For each request for recommendations of a user, RECOMM CANDIDATES determines a set of recommendation candidates. These are usually all items that have not been rated by the user. A PREDICT RATING operator uses the models to predict the rating score for each recommendation candidate resp. for each test tuple. It ensures a temporal and de-



a) Logical Query Plan b) Physical Query Plan

Figure 1: Logical and Physical Query Plan of a DSMS-based RecSys

terministic matching of models and recommendation candidates resp. test tuples. The RECOMMEND operator chooses the items that should be recommended (recommendation candidates with a minimal predicted rating and/or the top-K items).

The TEST PREDICTION operator implements an evaluation metric, e. g., RMSE. It compares the predicted and the true rating resp. the predicted and the true ranking position and aggregates an overall or moving average.

Physical Operators and Physical Query Plan

Figure 1b shows an example of a physical query plan as a result of the transformation from the logical plan. The left column implements the evaluation, the middle column the model training, and the right column the calculation of recommendations. Physical operators that belong to one logical operator of Figure 1a are surrounded by a dashed rectangle. Existing base operators are drawn with a solid line, new RecSys-specific operators (ITTT, BRISMF, RECOMM CAND, PREDICT RATING) with a dashed line.

The ACCESS operators at bottom of Figure 1b bind the input data streams and set the default validity intervals $[t, \infty)$. The SEND operators at top provides the output data streams. The TIME WINDOW operator next to the ACCESS operator of the requests for recommendations stream sets the validity intervals to $[t, t + 1)$. These requests are valid for the single point in time t.

After splitting the rating data stream into a test and learning stream by the ITTT operator (which implements the ITTT evaluation methodology as described below), the validity of the learning data is limited by a TIME WINDOW operator, so that each tuple is valid for a certain time span (e. g., for 6 months). The BRISMF operator is an implementation of the logical TRAIN RECSYS MODEL operator and implements the BRISMF algorithm [11]).

The models were joined with requests for recommendations by a CROSS PRODUCT operator. This operator joins elements with overlapping time intervals. Because we expect at every point in time exactly one valid model from BRISMF, each request for recommendations is joined deterministically with exactly one model. The RECOMM CAND operator outputs a stream element (u, i) as recommendation candidate for every item the requesting user has not rated yet. Each candidate is joined with a model, again with a CROSS PRODUCT operator. The PREDICT RATING operator uses the model to predict a rating $\hat{r}$ for the user-item pair of the candidate element and outputs it as predicted recommendation candidate $(u, i, \hat{r})$. A SELECT operator removes all candidates with a predicted rating less than $R_{\min}$ (e. g., $R_{\min} = 3.5$) and a TOP K operator creates a list of the top K items related to the predicted rating (e. g., $K = 8$).

The models for RECOMM CAND and for PREDICT RATING do not necessarily have to be the same. If the RecSys model does not allow to determine the unrated items of a user (e. g., user and item feature matrices as the result of matrix factorization), the model for RECOMM CAND can be a model that solely maps users to their unrated items. Therefore, we cannot pass the model through RECOMM CAND and need to join the candidates with the other models.

In this sample query, I use the ITTT evaluation methodology. At first, each rating tuple is used for testing and after that to train the model (cf. [9]). Because the scheduler of a DSMS controls which operator processes data and which one suspends, we cannot assure that a rating tuple is processed at first by the PREDICT RATING operator of the evaluation and after that by the BRISMF operator. However, with the help of the validity intervals, we can control which model is joined with the test tuple. The ITTT operator outputs for every rating tuple e two copies: a tuple e_T as test tuple and a tuple e_L as learning tuple. While e_L keeps the validity interval $[t_1, \infty)$ of e, the test tuple e_T gets a new validity interval $[t_1 - 1, t_1)$. That means e_T becomes invalid before e_L becomes valid. Due to the rules for the validity intervals of the models described in the last section, the validity interval of each model that used e_L for training starts not before t_1. Furthermore, the model that becomes invalid at t_1 (and is valid at $t_1 - 1$) did not use e_L for training. That means, that the model whose validity interval $[t_0, t_1)$ with $t_0 < t_1$ overlaps with the interval $[t_1 - 1, t_1)$ of e_T is exactly the latest model that has not used e_L for training.

Test tuples get a predicted rating by the PREDICT RATING operator. In general, this operator adds the predicted

rating $\hat{r}$ to the input tuple and removes the model, which was joined to the recommendation candidate resp. the test tuple. This means, for the recommendation task, PREDICT RATING outputs $(u, i, \hat{r})$ tuples and for the evaluation task $(u, i, r, \hat{r})$ tuples. Subsequently, the query calculates a moving root mean square error (RMSE) for the evaluation task. A MAP operator calculates for each test tuple the square error $se = (r - \hat{r})^2$. A TIME WINDOW operator defines the validity interval of the square error for the AGGREGATION operator: e. g., a time window that sets the validity of the square errors to 24 hours instructs the AGGREGATE operator to calculate for each new error value the mean value of all values of the past 24 hours. Subsequently, the MAP operator calculates the square root of the aggregated mean and hence outputs the RMSE for the past 24 hours.

Evaluation of Feasibility

I implemented this approach by extending the open-source DSMS *Odysseus*[1] [2]. Odysseus is designed to be modular and extendable. It supports an editor for the SQL-like stream-based query language CQL [3], the functional query language PQL [2], and a graphical query editor, as well as a dashboard to visualize the data and a GUI to control the DSMS. Additionally, developers can add new (domain specific) languages, new operators, new data source and sink connectors, and new dashboard parts by the use of the Odysseus framework.

I evaluated the feasibility of this approach by comparing the overall RMSE after each tested tuple for the MovieLens dataset with the results of the stream mining framework *Massive Online Analysis* (MOA) [5]. To make the results comparable, I added the MOA implementation of the BRISMF algorithm [11], removed the WINDOW operator that precedes BRISMF, and calculated the overall RMSE after each tested tuple with ITTT. The error values are exactly the same as those of MOA, which shows that the matching of learning data, models, and test data as well as the implementation of the evaluation operate correctly.

4. RELATED WORK

While a lot of research in the field of RecSys focuses on developing new online algorithms (e. g., [11]), our approach has the objective to incorporate these algorithms to a DSMS to benefit from its abilities for processing streaming data. Closely related is StreamRec [7]. It uses the stream processing system *Microsoft StreamInsight* to calculate item similarities with basic stream operators. In contrast to our approach, they do not present a generic usage of a DSMS as a RecSys. They show an implementation for item similarity.

There are some publications about stream-based systems that implement RecSys functions. For example, Ali et al. [1] propose an implementation of a distributed CF algorithm with *Apache Storm*. MOA [5] implements algorithms for stream-based RecSys. In contrast to our approach, they do not use an application-independent DSMS that processes stream elements with operators of a query plan with the advantages described above.

5. CONCLUSION AND FUTURE WORK

Our preliminary research on incorporating RecSys functions in a DSMS shows the feasibility as well as some challenges and prospects of this approach. It has the potential to benefit from well studied features of DSMSs. The future work focuses on the following aspects:

- Studying the implementation of other methods for recommending (e. g., other learning algorithms) and evaluation (e. g., ranking-based) with DSMS operators. This includes considering additional data sources, e. g., context data.
- Studying optimizations of the operator-based processing.
- Proving the suitability and flexibility of this approach in different settings.
- Evaluating and comparing of recommendation quality, latency, throughput, and memory usage in a real-world setting, e. g., in a Living Lab like Newsreel.[2]

Our approach allows to remove older ratings by the WINDOW operator. Other RecSys approaches use for example a gradient decay of learning tuples, a temporal bias [10], or a reservoir that holds samples of the learning data [8]. An important aspect of future work is to develop special implementations of the WINDOW operator to achieve these behaviours and to allow to compose these with different model learners.

6. REFERENCES

[1] M. Ali, C. C. Johnson, and A. K. Tang. Parallel collaborative filtering for streaming data. *University of Texas Austin, Tech. Rep*, 2011.

[2] H.-J. Appelrath, D. Geesen, M. Grawunder, T. Michelsen, and D. Nicklas. Odysseus: A highly customizable framework for creating efficient event stream management systems. In *DEBS'12*, pages 367–368. ACM, 2012.

[3] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *VLDB Journal*, 15(2):121–142, 2006.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS 2002*, pages 1–16. ACM, 2002.

[5] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive online analysis. *The Journal of Machine Learning Research*, 11:1601–1604, 2010.

[6] P. G. Campos, F. Díez, and I. Cantador. Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *UMUAI'14*, 24(1-2):67–119, 2014.

[7] B. Chandramouli, J. J. Levandoski, A. Eldawy, and M. F. Mokbel. StreamRec: A Real-time Recommender System. In *SIGMOD'11*, pages 1243–1246, 2011.

[8] E. Diaz-Aviles, L. Drumond, L. Schmidt-Thieme, and W. Nejdl. Real-Time Top-N Recommendation in Social Streams. In *ACM RecSys*, pages 59–66, 2012.

[9] J. Gama, I. Zliobaite, A. Biefet, M. Pechenizkiy, and A. Bouchachia. A Survey on Concept Drift Adaptation. *ACM Comp. Surveys*, 1(1), 2013.

[10] Y. Koren. Collaborative filtering with temporal dynamics. *Comm. of the ACM*, 53(4):89–97, 2010.

[11] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Scalable collaborative filtering approaches for large recommender systems. *The Journal of Machine Learning Research*, 10:623–656, 2009.

[1] `http://odysseus.informatik.uni-oldenburg.de/`

[2] `http://www.clef-neewsreel.org/`