StreamER: Evaluation Framework For Streaming Recommender Systems

By
Sai Sri Kosaraju

Supervisors:
Dimitris Paraschakis
Bengt J.Nilsson

Examiner:
Johan Holmgren
Contact information

Author:
Sai Sri Kosaraju
E-mail: saisri.kosaraju@gmail.com

Supervisors:
Dimitris Paraschakis
E-mail: dimitris.paraschakis@mau.se
Malmö University, Department of Computer Science and Media Technology

Bengt J.Nilsson
E-mail: bengt.nilsson.ts@mau.se
Malmö University, Department of Computer Science and Media Technology

Examiner:
Johan Holmgren
E-mail: johan.holmgren@mah.se
Malmö University, Department of Computer Science and Media Technology
StreamER: Evaluation Framework For Streaming Recommender Systems

Contents

1 Introduction 12
  1.1 Motivation ......................................... 14
  1.2 Research Goal .................................... 16

2 Research Methodology 17
  2.1 Design Science .................................... 17
  2.2 Research Phases .................................. 20

3 Literature Study 22
  3.1 Evolution of Recommender systems .......... 22
  3.2 Classification of Recommender systems .... 23
  3.3 Streaming Recommender Systems ............ 25
  3.4 Recommender Systems Metrics ............... 29
  3.5 Tool Kits and Other Libraries ............... 30
  3.6 Comparative analysis ............................ 31

4 Design of The Evaluation Framework 33
  4.1 Design Goals ..................................... 33
  4.2 Design Overview ................................ 34
  4.3 Data ............................................ 37
  4.4 Evaluator ....................................... 39
  4.5 Algorithms ..................................... 43
  4.6 Communication Protocol and Event Streaming 43
# StreamER: Evaluation Framework For Streaming Recommender Systems

## 5 Implementation

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Input Data</td>
<td>45</td>
</tr>
<tr>
<td>5.2 Evaluator</td>
<td>48</td>
</tr>
<tr>
<td>5.3 Algorithms</td>
<td>51</td>
</tr>
<tr>
<td>5.4 Communication Protocol</td>
<td>53</td>
</tr>
</tbody>
</table>

## 6 Evaluation and Results

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Results</td>
<td>55</td>
</tr>
<tr>
<td>6.2 Evaluation Discussion</td>
<td>58</td>
</tr>
<tr>
<td>6.3 Limitations</td>
<td>60</td>
</tr>
</tbody>
</table>

## 7 Conclusions and Future Work

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1 Conclusion</td>
<td>62</td>
</tr>
<tr>
<td>7.2 Future Work</td>
<td>63</td>
</tr>
</tbody>
</table>
StreamER: Evaluation Framework For Streaming Recommender Systems

Abstract

Recommender systems have gained a lot of popularity in recent times due to their application in the wide range of fields. Recommender systems are intended to support users in finding the relevant items based on their interests and preferences. Recommender algorithms proposed by researchers evolved over time from simple matching recommendations to machine learning algorithms. One such class of algorithms with increasing focus is on called streaming recommender systems, these algorithms treat input data as a stream of events and make recommendations. To evaluate the algorithms that work with continuous data streams, stream-based evaluation techniques are needed. So far, less interest is shown in the research so far on the evaluation of recommender systems in streaming environments.

In this thesis, a simple evaluation framework named StreamER that evaluates recommender algorithms that work on streaming data is proposed. StreamER is intended for the rapid prototyping and evaluation of incremental algorithms. StreamER is designed and implemented using object-oriented architecture to make it more flexible and expandable. StreamER can be configured via a configuration file, which can configure algorithms, metrics and other properties individually. StreamER has inbuilt support for calculating accuracy metrics, namely click-through rate, precision, and recall. The popular-seller and random recommender are two algorithms supported out of the box with StreamER. Evaluation of StreamER is performed via a combination of hypothesis and manual evaluation. Results have matched the proposed hypothesis, thereby successfully evaluating the proposed framework StreamER.
Popular science summary

Recommender Systems are the collection of technologies that process, filter and classify information to provide users with recommendations. The recommender system facilitates users in making choices without sufficient personal experience of the possible alternatives. Class of recommender algorithms that operate on continuous data is called streaming recommender algorithms. These systems aim to solve the complex problem of producing recommendations for a customer in real time while processing a continuous stream of data.

Any researcher developing a new algorithm requires to demonstrate that their algorithm is accurate and it performs better when compared to similar algorithms in the past. To do so an evaluation of the algorithms is required. The evaluation is testing the system for its accuracy to produce relevant recommendations.

There is a gap in the research of offline evaluators that can handle streaming recommender systems. This thesis aims to cover that gap by providing an offline evaluator for streaming recommender systems. The framework developed as part of the thesis is called as StreamER. It should support any algorithm, metric or reward mechanism. Recommender system metrics can measure various aspects of algorithms, such as accuracy, diversity of recommendations.

StreamER can be described by its internal modules. Data module is the first of them, it contains configuration and input data. Configuration data is used to drive the behavior of the StreamER. Since StreamER is a generic framework, it should support any variant of streaming data for recommender systems. This poses a challenge that they can have different formats and
StreamER: Evaluation Framework For Streaming Recommender Systems

different fields. To make it easy for implementation, StreamER describes its own input data format and has an inbuilt support for converting different data formats to the native StreamER format. All this is part of the data module. As part of thesis implementation, yoochoose dataset is used. It is an e-commerce dataset with the click and buys information from various items and users sorted with increasing timestamps.

The second module of StreamER is an evaluator, it consists of modules to calculate metrics and rewards. Metrics to track the performance of algorithms while rewards provide feedback to algorithms about the quality of recommendations. In this thesis click-through rate, precision and recall metrics are implemented to evaluate the accuracy of algorithms.

Next module consists of recommender algorithms providing recommendations. In this thesis, the random recommender algorithm and popular-seller recommender algorithm are implemented. The random algorithm will provide random items as recommendations for the given list of items. The popular-seller recommender algorithm provides top purchased items so far as recommendations. Finally, communication between all the modules is handled via a communication protocol that has predetermined data packet formats.

StreamER evaluation process starts with reading the first event in the event data file by the evaluator and sending it to algorithms followed by request for recommendations. The evaluator waits until all the algorithms have provided recommendations. Once recommendations are available, the evaluator provides rewards to algorithms for their successful recommendations and calculates metrics. Finally, the evaluator checks for the next event and sends it to the algorithms. The process repeats until all the events are exhausted. It is important to note that the evaluator sends one event at a time to algorithms.

A hypothesis is proposed for the evaluation of StreamER. The results of the calculated metrics are plotted in the graphs. It can be seen that popular-seller recommender algorithm performs better than the random recommender algorithm. This is because the random recommender algorithm provides random recommendations. While popular-seller algorithm has a proven method
StreamER: Evaluation Framework For Streaming Recommender Systems

behind its recommendations. By analysis of results both from hypothesis and manual inspection, StreamER is evaluated to perform as expected.
Acknowledgement

I am grateful to my supervisors Dimitris Paraschakis and Bengt J.Nilsson for their co-operation and support throughout the thesis. I would like to thank them for helping me with all the ideas about Evaluation Framework. I would also like to thank my examiner Johan for the support and guidance throughout the thesis.
List of Figures

4.1 Design Overview of the framework .................................. 35
4.2 Evaluation Framework Process flow chart ............................ 36
4.3 Types of Messages ...................................................... 44
5.1 Algorithm work-flow .................................................. 52
6.1 Precision .............................................................. 56
6.2 Click Through Rate .................................................... 57
6.3 Recall ................................................................. 57
6.4 Rewards ............................................................... 59
List of Tables

4.1 Configuration File Fields ........................................... 38
4.2 Event Data File Fields ............................................. 39
4.3 List of Metrics ....................................................... 41
5.1 Metric Base Class Functions ......................................... 49
5.2 Abstract functions of Algorithm .................................... 53
List of Acronyms

CPU  Central Processing Unit
CTR  Click-through Rate
DS   Design Science
GPU  Graphics Processing Unit
OO   Object Oriented
OOP  Object Oriented Programming
RS   Recommender Systems
SR   Streaming Recommenders
Chapter 1

Introduction

Watching a video on YouTube, a movie on Netflix, buying a product on Amazon, selecting a song in Spotify, are some of our regular online interactions. All these examples have two common attributes, Choice and Recommendation. Interaction usually starts with choosing one item from the given list of items and this triggers recommendations from the website. This process is handled by a set of systems that turn simple interactions into a recommendation output. These systems are known as Recommender Systems and have become dominant in daily online interactions.

Recommender Systems are the collection of technologies that process, filter and classify information to provide users with recommendations. The recommender system facilitates users in making choices without sufficient personal experience of the possible alternatives [20].

To understand recommender systems a step back in time before the advent of the internet era is necessary. Sources of recommendations in the world before the information technology revolution were word of mouth, printed guides, reviews in magazines etc [20]. People used the information presented from these sources to make their choice.

Coming back to today’s trend, online shopping in recent years has presented users/customers with the copious amount of information, products, and services. This data explosion leads to disarray and exhaustion of customers when consuming information or products [9].
StreamER: Evaluation Framework For Streaming Recommender Systems

To handle this information management problem, recommender systems have been developed. In the last decade, many recommender algorithms have been deployed in e-commerce and streaming services. Traditional systems matched items with users based on feedback and rating provided by users. Latest recommender algorithms utilize more sophisticated techniques such as machine learning and data mining. With the exponential growth of information in e-commerce, streaming, social network domains, recommender systems are and will be the de-facto way of item discovery.

Recommender systems can be seen as consisting of two different components: algorithms and data. Algorithms provide recommendations from the data provided to them. Classic recommender algorithms were driven by data generated from item description and user feedback called as "Explicit Feedback/Explicit Data". Algorithms in classic recommender systems used the explicit feedback to suggest recommendations. This process of recommendations seems trivial with algorithms acting on an explicit-data to provide recommendations.

Online services today can track every movement of their users during the interaction. Tracking of these interactions can provide a huge amount of data known as "Implicit Data". It consists of various actions performed by the user in a given session or over a period of time. Observing a purchase process provides an understanding of the implicit data generation.

The process of buying a product starts with searching/browsing through e-commerce sites like Amazon or e-bay. This action tells the website that the user is interested in a product, or category. The search usually is followed by user browsing through the displayed results, adding one of the items to cart and in the end purchasing. From this transaction, online service knows that a user is interested in a category of items etc. The data captured in this process is implied from user actions rather than explicit feedback. To an untrained eye, data generated in such fashion looks complex and random. But Modern recommender systems can work on both implicit and explicit data and thereby provide valuable recommendations.

Extending the above-described process to millions of users performing actions will result in gigabytes of data. At this point, data cannot be seen
as a discrete set of events, but instead as a continuous stream, each set with its own actions and timestamps. This explains the streaming nature of the data in recommender systems [4]. Algorithms operating on continuous data are called streaming recommender algorithms. These systems aim to solve the complex problem of producing recommendations for a customer in real time while processing a continuous stream of data.

1.1 Motivation

So far it is established that the recommender system algorithm development is happening at a rapid pace. The fruits of research in this area are evident from the seamless working of everyday e-commerce sites. More people using online services mean more important recommender system development is. This result in the increase of research on recommender algorithms is evolving thus producing new techniques and methods to handle data. Along with this rapid pace of algorithm development comes the requirement of demonstrating their performance and correctness. Any researcher developing a new algorithm requires to demonstrate that their algorithm performs better when compared to similar algorithms in the past. In simple terms evaluation of the algorithms also became important. The evaluation of an algorithm refers measurement of its ability to produce expected results, along with benchmarking algorithm’s performance and any other metrics required.

A review of state-of-art research in developing evaluation metrics, benchmarks and systems shows very little promise. Researchers have proposed a few evaluation libraries but, most of them are not designed to serve streaming recommender algorithms. Usually, the algorithms proposed comes with targeted evaluations. This means new researchers are on their own for evaluation when developing new algorithms.

Among existing libraries, only a few are capable of handling the streaming data, such as Idomaar [11], ScaR [12], Prequential evaluation protocol [24]. Idomaar used in [11] is open source and extendable evaluation framework but, the learning curve is quite steep and it is a complex system. The prototype of the prequential evaluation protocol is not available for the public use
StreamER: Evaluation Framework For Streaming Recommender Systems

[24]. ScaR [12] is an online framework written in Java, thereby prompting developers to stick to that language. The aim of ScaR is to provide collaborating services that can produce recommender algorithms. Finally, all these libraries are quite heavy and favor large-scale deployment.

From the above discussion, it is clear that there is a gap in offline evaluators that can handle streaming recommender systems. This thesis aims to cover that gap by providing an offline evaluator for streaming recommender systems. From the discussion, it is also clear that the framework should be flexible, easy to use and also open sourced. Flexibility is required because users might want to compare different algorithms, their attributes, metrics etc. A flexible design will allow users to do the same without having to rewrite every time a different benchmark is needed. Easy to use is important because the main aim of these researchers is developing algorithms and not spending time on building/adapting evaluation framework(s). For example, the yearly recsys challenge aims to solve a particular problem in recommender systems [1]. When competing for such challenges, it is important to spend efforts in creating better algorithms.

The contribution of this thesis will be made by designing and implementing the simple python-based evaluation framework for recommender algorithms. In particular, this thesis will focus on the framework for streaming recommender algorithms. This framework is intended for the rapid prototyping of incremental algorithms and standard metrics are provided for evaluation of algorithms. Name of the framework is StreamER (stream- streaming data, E- evaluation, R- recommendations). Any researcher or organization can use this framework to evaluate streaming recommender algorithm(s) of their choice. The user of the framework can plug-in their algorithm and communicate with the evaluator via a communication protocol. The evaluator will evaluate the algorithms for the accuracy of the recommendations produced by them.

[1]https://recsys.acm.org/recsys18/challenge/
1.2 Research Goal

A research goal is described below to arrive at the proposal made in motivation. Research goal will be achieved by answering the proposed research questions.

Research Goal - Design and Implementation of an Evaluation Framework for streaming recommender Systems

- \textit{RQ: How to build an off-line evaluation framework for streaming recommender systems?}
  - How to design the evaluator that is flexible, and easy to use?
  - How to design a communication protocol between the evaluator and the recommender system algorithm in such a system?

With motivation and research goal established, Chapter 2 describes the research design and methodology used. The current state of the art in recommender systems is presented in Chapter 3. Design and implementation of the framework are explained in detail in Chapters 4 and 5 respectively.
Chapter 2

Research Methodology

In this chapter, the research methodology selected to answer the research goal and its suitability for the research is discussed. The first section contains the general overview of the design science methodology and its suitability to the research, followed by the description of the steps undertaken in the research.

2.1 Design Science

Design science (DS) methodology is chosen as the research methodology to address the research questions proposed. According to Henver et al. [10] design science methodology is a problem-solving paradigm. This deals with the creation of new knowledge by developing artifacts to solve the potential problems via analysis, design, implementation, and evaluation [10]. The problem-solving attribute of the design science made it suitable for this thesis, while the shortage of evaluating frameworks for stream-based recommender system algorithms is the problem being solved.

March and Smith [15] proposed that artifacts are outcomes produced by the design. Build and evaluate are important activities that should be included while designing the artifacts. The authors classified them into 4 types, namely Constructs, Models, Methods, and Instantiations. Artifacts are expected to make a unique contribution to existing knowledge in order to be acceptable. An artifact of this thesis, namely "evaluation framework-
StreamER: Evaluation Framework For Streaming Recommender Systems

StreamER” will be developed in the form of an instantiation. Instantiation is nothing but the realization of an artifact in its real environment. It aims to provide the practical representation of artifacts. Instantiation also provides the presentation of effectiveness and feasibility of artifacts that it incorporated. Research activities include representing the needs of potential users, transforming needs into the system specific requirements and finally transforming those requirements into a working system by implementation [15].

As part of this thesis, the lack of sufficient evaluation systems for stream-based algorithms is identified as a gap in the existing research. Design of an evaluation framework with its features is proposed and converted into a working prototype by the implementation.

According to Hevner et al [10] there are seven guidelines for a project using Design Science Research. Suitability of the design science methodology to the thesis is explained clearly below with respect to the guidelines.

- **Artifact**: The first guideline for a project using DS is that it must produce an artifact in the form of an algorithm, model, framework etc. Thesis produces an evaluation framework in the form of an artifact for evaluating the streaming recommender systems. Implementation of the framework and its evaluation corresponds to build and evaluate activities of the research method.

- **Relevance**: The second guideline of DS methodology is to have a clear relevance to business problems. Project goal and motivation clearly satisfy the relevance aspect of the methodology.

- **Evaluation**: Evaluation of design is a major component of the design science. In this thesis, the implemented framework is evaluated based on functionality. Evaluator functionality is to provide streaming data to algorithms while also generating the metrics and rewards. Rewards are the binary indication of the successful recommendation made by algorithms. A complete picture of evaluation would constitute of verifying evaluator handling streaming data, providing rewards and calculating metrics. To support with all these steps manual verification and a proposed hypothesis in Section 2.2 are used.
StreamER: Evaluation Framework For Streaming Recommender Systems

- **Contribution**: Evaluation framework developed is the main deliverable of this thesis. It can be used as an evaluator for any streaming data based recommender algorithm.

- **Rigor**: Evaluation framework selected for implementation is extensively researched and is implemented using Python. A protocol is designed for the communication between the recommender system algorithm and evaluation framework.

- **Search Process**: An extensive survey of the literature is conducted. The search for articles on the topic is executed carefully in every step starting from search terms to the reading of the selected articles as presented in Section 2.2.

- **Communication of Research**: Finally the communication of the research of this thesis will be done by publishing the report.

Before selecting design science as a methodology, qualitative, quantitative approaches are considered. Qualitative methods are firmly based on the inclusion of surveys from various sources for the input data [17]. Even though basic recommender system feedback can be considered as a kind of survey, implicit feedback data used in recommender systems cannot be counted as a survey data. Other qualitative research methods such as ethnography, the case study can be partially applicable to the thesis. Ethnography deals with the study of people and cultures. Though recommender systems might be tuned based on people and culture they are aiming to serve, it does not encompass the whole recommendation process. In particular, ethnography has no hold over evaluation, since the process is same across the board. Case study methodology deals with the real-life study of specific examples of complex phenomenon [10]. This thesis does not deal with a specific case, instead implements a generic framework for evaluation.

The quantitative approach also seems partially applicable to the thesis as it involves the development of scientific models and methods for measurements. The mathematical nature of the recommender systems and the evaluation framework also work towards the quantitative approaches. For
quantitative methods, variables need to be clearly defined. Finally, measurement data and theory in the quantitative research must be deterministic \cite{17}. The determinism of input data is true for this thesis since input data is from a recorded session. It is clear from the above discussion that DS is the suitable method for this thesis.

## 2.2 Research Phases

The flow of research process undertaken in this thesis project constitute the following phases and each phase is iterated to arrive at solutions.

- **Literature Study:** Literature study is conducted extensively to gain the in-depth understanding of current state-of-art techniques and previous research done in the field of recommender systems. This helps to understand the gap in previous research, thereby contributing to the motivation of the thesis. Understanding the gap was important in order to make a unique contribution to the existing knowledge. Comparative analysis is also provided to differentiate the contribution of existing work on the evaluation of recommender systems.

  Major keywords used while searching for the literature are ”recommender systems”, ”evaluation of recommender systems”, ”metrics”, ”click-through rate”, ”prequential evaluation”, ”stream-based recommendations”, ”benchmarking recommender systems”, ”evaluation tool for recommender systems” etc. Filtering of relevant literature from the found information is done by reading the abstract and introduction. All the relevant literature is studied carefully after initial filtering.

- **Design:** In design phase all the findings from the literature study are gathered and analyzed in order to design the prototype. The prototype is designed from scratch and design features of the prototype are finalized after many iterations. Sub-components and their corresponding functionality are decided in this phase.

- **Implementation:** Designed prototype is implemented in this phase
StreamER: Evaluation Framework For Streaming Recommender Systems

of research. The prototype of the evaluation framework developed is expected to provide the proof of concept for the research done in this thesis.

- Evaluation: Streaming recommender system evaluator functionality includes handling of input data, communicating data to the algorithms, requesting recommendations, providing rewards and calculating metrics for the recommendations received. This functionality can be evaluated for its accuracy and correctness. The evaluator is said to be functionally correct when it can do all the steps as required. At the end of the evaluation, the output will be generated and collected in different formats. Output in this thesis can be in the form of plots and be writing to files.

Evaluation of artifact for correctness will be accomplished by proposing and proving a hypothesis. The hypothesis proposed is "This thesis will select two different algorithms to implement and measure selected accuracy metrics along with rewards provided to them. Algorithms for implementation will be selected in a way that one will perform much better than the other algorithm in every sense. This performance difference will be observable in the final output. The difference in the performance of algorithms will be explained by their underlying mechanism of generating recommendations."

Satisfying functional correctness and accuracy hypothesis will thus evaluate the artifact of this thesis.
Chapter 3

Literature Study

The literature study of this thesis is organized to build from the basics of recommender systems to state-of-art evaluation techniques. First two sections of this chapter detail the evolution and classification respectively, followed by analysis of streaming recommender systems. The next section discusses evaluation metrics and techniques. The final section presents various recommender system libraries and toolkits available.

3.1 Evolution of Recommender systems

One of the first articles describing recommender systems is by Goldberg [8]. It outlined collaborative filtering (CF) as a solution for handling incoming email and documents using a system called Tapestry. It was both an email filter and storage solution, people collaborated to perform filtering. Collaboration consisted of providing reactions to the information users received, such as emails or document. This was a novel idea at that time and formed the basis for the subsequent research in the area. Resnick et al [20] coined the term "recommender system" and identified the fact that providing feedback may not explicitly collaborate with the users. This is due to the fact recipients of the recommender systems may not belong to the same organization, location. The paper also proposed that recommendation may also suggest interesting items instead of just filtering. This was a major leap from the simple idea in
StreamER: Evaluation Framework For Streaming Recommender Systems

Soon the advent of online shopping sites starting with Amazon in 1994 and the dotcom boom of the late 90s have contributed to the evolution of recommender systems.

3.2 Classification of Recommender systems

Recommender systems are classified into three main categories in the classic paper [1] by Adomavicius. The first being Content Based (CB) Filtering. In this, the system recommends items that are similar to the one(s) the user has liked/used in the past. The other is Collaborative filtering (CF) in which feedback collected from users is used to suggest items to other users with similar behavior. In the paper, they also propose a system that combines both CB and CF into a hybrid system, the third category of the recommender systems.

CB filtering systems provide recommendations based on the contents of items being recommended. CB systems recommendations are built on the foundation of user feedback that is gathered from surveys or reviews. All future recommendations are provided by finding items similar to the one(s) that have received good feedback or review. Finding items based on similarity requires a detailed description of all items in the database, which can pose challenges with the increase of items or missing description. A system is said to have a cold-start with respect to an item or user when there is no background data associated with them. [1] and [9] have identified cold-start as the biggest challenge for a CB recommender system.

CF filtering systems derive recommendations by suggesting the items consumed by the user with similar behavior. Algorithms in CF find the nearest neighbors that are behaviorally alike. CF also suffers from the cold start problem when a new user is added and no nearest neighbors are yet calculated. When a user is unique in tastes with no other user with similar tastes, it becomes difficult to provide recommendations. Unique users pose a challenge as there is a chance that a system has few unique users and/or the number of users on the system is too small to have users with similar behavior [1].
Among CF techniques, Matrix Factorization is by far the most popular as mentioned by Guillou [9]. These techniques construct a matrix where users and items form the rows and columns, the known data is filled and the idea is to predict the rest of the matrix. A model is built that uses user ratings to make the future recommendations instead of just making predictions. One of the matrix factorization methods proposed by Bauer et al [2] introduces a model based on implicit customer feedback that is derived from the transactions and other information during a customer interaction with the system. The same paper proposes solutions to handle the sparsity problem where the number of users and items become so huge that the matrix is sparsely filled. They also propose a method to handle the skewness of implicit feedback. Drawbacks of the methods by Bauer include monotonous suggestions and lack of item interaction sequence understanding.

Session-based recommender systems are a new class of recommender systems that handle the information in the form of individual sessions to provide recommendations. Session-based Recommender systems use techniques such as language prediction models. Usually, these prediction models use either traditional modeling methods such as Markov chains or neural networks. [25] by Wu et al was the first paper that suggested the use of language prediction models to solve the session based recommender systems. The Recurrent Neural Network (RNN) has two parts where the recurrent part handles the historical feedback while non-recurrent use preferences. Another paper using RNN [5] by Devoogt et al proposed using RNN to provide both short and long-term recommendations using session-based collaborative filtering.

So far we have discussed various types of recommender systems. The exponential growth of users and products have rendered above methods less useful since they are not designed to support the incremental updates [3]. Item and user growth, mixed with lack of user profiles has given rise to a new class of recommender systems based on streaming data. These recommender systems use the stream of data to provide recommendations in real time to the user. The next section presents state-of-art in stream-based recommender systems.
3.3 Streaming Recommender Systems

3.3.1 Introduction

The increase of data, transactions, and continuity of the data flow has given rise to the concept of streaming recommender systems in [22]. A simple example would be a site like eBay where thousands of transactions happen every second while the items and users keep increasing.

The first step in understanding streaming recommender systems is to understand the properties of streaming data. Chang et al [4] has identified high-speed, variable size and changes in product landscape as some of the inherent properties of streaming data. Andreas and Sahin [13] describe short time to live as an important attribute of the streaming data. One common theme among all the research articles is the huge scale of streaming data.

Nature of streaming data discussed above shows the glaring weakness of traditional algorithms that assume static data set. Chang et al [4] proposes a recommender system sRec that can handle the dynamic creation and deletion of users/items while allowing the changing of ratings. The input data is modeled as feedback activities, new users, and new items. The algorithm proposed utilizes a random process that can handle the streaming data. During the evaluation of algorithm cold start is effectively modeled and tested, but adding and removing new users or items is not tested.

3.3.2 Related work

Lommatzsch et al [13] suggests that the streaming nature of data demands some requirements such as providing recommendations within a certain period of time. They argue that the lifespan of items is very short, especially in streaming environments due to the reason that the relevance of the items changes very quickly. Considering the requirement authors analyzed the nature of streams in online news portals and based on the findings of the analysis, proposed approaches to provide the recommendation in the field of

\[^2\text{https://www.webretailer.com/articles/ebay-statistics.asp}\]
online news streaming environments.

The authors discussed the PESTA contest whose objective is to provide an opportunity to evaluate the stream base recommender algorithms in real-world settings where the data stream is dynamic. The authors tried to observe the data streams used in PESTA contest in order to understand the characteristics of data streams. These observations gave an insight that the streams tend to differ based on items lifespan, number of items, the popularity of the items and the context. In the contest, the recommender algorithms are evaluated for their performance using the click-through rate. In order to compare various algorithms in parallel measures such as near-to-online precision are used. Click through rate is nothing but the proportion of clicked recommendations to the total number of recommendation lists while near-to-online precision focuses on measuring the precision of the recommender algorithm to predict the clicks of the user. Evaluation of the various algorithms made it evident that the performance of the algorithm is dependent on the domain and context. Since it is impossible to find the algorithm that works in all contexts, the authors have combined various algorithm scenarios and navigated the recommendation requests to the best suitable algorithm. This process managed to produce the high-quality recommendations and stood the best in the PESTA contest.

Lacic et al [12] proposed ScaR, a micro-service architecture based recommender framework that can handle the large streams of data and provide recommendations in real time. It is open source, Java-based and relies on Apache solr and Apache Zookeeper. The framework is implemented in a very scalable way to facilitate the handling of various algorithms and large-scale dynamic data streams. The main principle used in the implementation of this framework is the modularization of components. Each component in this framework is an independent service that can operate on its own and communicates with the other components with the help of lightweight mechanisms. They efficiently made use of the features of Apache solr to handle the data streams. The performance of the models has been evaluated with both online and offline evaluation methods using the framework.

Chandramouli et al [3] discussed the importance of providing real-time
recommendations in recommender systems, especially in settings where the system is subject to frequent updates. In such cases, the accuracy and quality of the recommendations produced by the system are highly affected. They also highlighted that most of the state of the art recommender algorithms lack the technology to handle the real-time recommendations. In order to address the flaws mentioned authors proposed an architecture named StreamRec that incorporates the ability to process the data streams in order to provide real-time recommendations. Using StreamRec one can implement the recommender systems in the form of an event processing applications. In this architecture, the user subscribes to the events from the recommender engine. The recommender engine processes the recommendation requests and provides top n recommendations to the user. Any update in the recommendation list will be notified to the user, thereby ensuring that the user receives relevant and real-time recommendations.

Loni et al [14] mentioned limitation of storage space in applications in order to perform memory resident operations and need for computational capability as the main challenges faced in providing real-time recommendations.

Karthik et al [23] describes the challenges of traditional algorithms in the streaming scenario. Those are the requirement of offline phase, factorization of a matrix of entries, and temporal nature. They have proposed an offline neighborhood-based model that is then used for streaming data sets via a min-hash approach.

3.3.3 Research in Evaluation of Streaming Recommender Systems

Kille et al [11] discussed NEWSREEL lab that is an evaluation framework for stream-based news recommender systems. This lab’s objective is to evaluate the various news recommender algorithms that work on streaming data both online and offline. In the news recommendation scenario the user and item/article data sets are very dynamic in nature since the news articles are considered relevant until a certain period of time and after that, the article
StreamER: Evaluation Framework For Streaming Recommender Systems

needs to be replaced by the new ones. Due to this requirement, the online news recommender system often seems to face the cold start and lack of user profile challenges. These challenges of news recommender algorithms are addressed by NEWSREEL lab using online and offline evaluation. The online evaluation measures the performance of recommender algorithms using metrics such as click-through rate, while offline evaluation focuses on analyzing recommendation precision and technical complexity of different algorithms. The sole objective of the online evaluation is to boost the click-through rate of recommended items. In the offline evaluation, unlike the online one the data does not directly come in live, instead the data stream is recorded in the online scenario and simulated back in the exact same way. In such situation, the click-through rate is computed on delay basis while it was calculated in the online scenario immediately after the recommendation is made. In order to perform this offline evaluation author proposed a framework called Idomaar which is platform and programming language independent. In NEWSREEL 2015 challenge 24 countries took part and tested their algorithms.

In [24] authors argue that the accuracy results of the recommender algorithms those are evaluated in controlled environments such as laboratories may not be directly adapted to the real world setting. This is due to the fact that in the controlled environment the evaluation is done using the datasets with static data while in the real world the data is continuous. The paper also discussed the batch evaluation and issues associated with it. Those are issues such as problems associated with shuffling datasets, and rearranging datasets using sessions and users might be expensive. The detailed description of batch evaluation is presented in the Section 3.4. An evaluation protocol namely prequential evaluation protocol is proposed by the authors that can be used to evaluate the recommender algorithms both in the real world and controlled settings. This evaluation protocol is specially designed to operate on the continuous data streams. This protocol allows the continuous observation of performance measures at the individual session level to entire dataset level. It also provides a way to continuously add the user data in a loop without stopping at a particular data point. To do so, it uses the test and learn scenario where whenever a new data point is found recom-
recommendations will be made and tested. The model/algorithm will be updated using the new data point making this framework suitable for the streaming environments. The protocol mainly focuses on the evaluation of algorithm efficiency to predict the next item. Three different incremental algorithms are tested using this protocol and results shows that the protocol is considerably good at close evaluation of algorithms.

With the above research in recommender systems, it is evident that many algorithms are proposed, but stream-based evaluation mechanisms are often overlooked. In the next section list of evaluation systems and metrics currently used are presented.

### 3.4 Recommender Systems Metrics

Evaluation of traditional recommender systems is done using a batch evaluation technique. In the batch evaluation process, a dataset is divided into training and testing sets. In the training phase, the training dataset is fed to the algorithms to train them. Once the algorithm is trained, it is tested using the testing dataset. Recommendations from the test phase are evaluated using various metrics. These metrics measure the attributes of the algorithm such as accuracy, diversity etc. The accuracy measures the error between the given and predicted ratings.

On the other hand, while evaluating the stream-based recommender system batch evaluation technique is not used. The algorithm will be trained and tested using the continuous stream of data and the recommendations produced by the algorithm are evaluated using various metrics.

The following list shows the few of accuracy metrics used for evaluation of recommender systems.

- **Precision**: Measures the fraction of relevant recommendations to the number of recommendations.

- **Click Through rate (CTR)**: Measures the ratio of clicks to the number of recommendation lists provided.
Recall: Measures the ratio of relevant items recommended to the total relevant items.

[5] by Devoot et al uses metrics that measure the accuracy of recommendations in session-based recommender systems. They used the recall and precision metrics. In an article McNee et al [16] present arguments against the focus on accuracy metrics in recommender systems. According to the paper, accuracy metrics focus too much on providing accurate individual items. This process results in recommendation list that is too narrow and homogeneous. Radlanski et al [19] used Click Through Rate (CTR) and a fraction of users with the relevant document as metrics for measuring the performance of their algorithms using multi-armed bandits. The fraction of users with relevant document measures the number of users who received a relevant document for the given recommendations.

Bias in the recommender systems can be described in terms of the user model and the item selection model. When the recommender system shows bias towards a specific user, it might end up suggesting too similar recommendations. For example, If the recommender system models the user as a geek, it might keep suggesting sci-fi gadgets or similar items. In reality, the user could be a normal user who at some point bought items that triggered the bias. When it comes to item selection bias, the recommender system might try to be conservative and suggest items that are too close to the original purchases thereby slowly biasing towards a class of items. For example, a user might buy a microwave on a Black Friday sale and the recommender system might keep suggesting ovens and other kitchen items. In fact on a sale, the user might expect other items with huge discounts in every department.

3.5 Tool Kits and Other Libraries

Devooght et al [5] proposed a Python library that contains various CF algorithms such as Fossil, Markov chains, Recurrent neural networks etc. All of these algorithms use session data of the user from Movielens, Netflix and Yoochoose data sets. Session data is split into 3 data sets namely Train-
StreamER: Evaluation Framework For Streaming Recommender Systems

ing, Test and Validation datasets for the evaluation process. The evaluation process is carried out by calling scripts from a command line interface.

Rival is another well-known evaluation toolkit implemented using Java [21]. The evaluation process consists of four phases in which data splitting, item recommendation, candidate item generation, and performance measurement are done. Rival is a toolkit for evaluation, the process of item recommendation in the second phase of evaluation is done by the recommender framework that is under evaluation. A cross-platform comparison of recommender frameworks is done using various evaluation strategies.

MyMediaLite [7] is a multi-purpose recommender system library that allows the user to use the existing algorithms for implementation and evaluation purposes. It is a C# based, open source library and supports the reusability of an existing algorithm instead of implementing them repeatedly, which can save a lot of time and effort. This library does the rating prediction and items prediction in collaborative filtering. In order to make item predictions only positive user feedback is considered. On the other hand to make rating predictions both positive and negative feedback is considered.

Lenskit [6] is another famous recommender systems toolkit which uses collaborative filtering algorithms. This toolkit provides implementations for collaborative filtering algorithms and APIs for most common use cases in recommender systems. It also allows the users to perform the offline evaluation of the recommender algorithms. The sole purpose of this framework is to provide the flexible, reusable environment to the user where different algorithms can be implemented, compared and evaluated easily.

3.6 Comparative analysis

Above discussed libraries use the batch evaluation technique, where discrete data from the datasets will be split into sections or batches. Some batches of data will be used to train the algorithms, while the other batches are used for evaluation. All of them are not designed to handle the streaming data. The evaluation framework developed as part of this thesis will make a different contribution by evaluating the stream of continuous data. This
StreamER: Evaluation Framework For Streaming Recommender Systems

framework will not use the batch evaluation technique. Instead, it uses the entire data to train and evaluate the algorithm. Recommendations produced by the algorithm are evaluated using metrics such as CTR and precision.

However, evaluation systems like prequential evaluation protocol, Idomaar[^3] and ScaR[^4] are designed for streaming data. But no prototype of the prequential evaluation protocol is available to use. The prototype of the framework implemented in this thesis is an open source tool. Idomaar is the well-known framework which is an evaluation system that is developed to evaluate the news recommender systems. The ScaR is the framework that is intended to work with the large-scale systems while, the StreamER prototype developed in this thesis will be lightweight with minimal functionalities and can be used for the rapid prototyping of incremental algorithms. For the evaluation of algorithms set of few accuracy metrics is used. The available evaluation systems that handle streaming data have complex architecture. While this thesis produces an evaluation framework that is so simple that one can easily plug-in the algorithms and evaluate them. The framework is implemented in python which is widely used in the implementation of recommender system algorithms.

[^3]: https://github.com/crowdrec/idomaar
[^4]: http://scar.know-center.tugraz.at/index.html
Chapter 4

Design of The Evaluation Framework

4.1 Design Goals

Design of StreamER is the first step in achieving research goal of this thesis. Before starting with design, it is important to understand input data and remember the motivation for StreamER. Streaming data is unique as it does not have a predetermined length. This, in turn, impacts design in terms of memory, data path construction and data processing. Data path can be defined as the series of steps each input data item travels through to generate the required output. Data path and processing for handling streaming data must follow a sequential path and must handle only one item in the sequence at any given time instance. Finally, memory management must be both static and dynamic with allocations and freeing happening wherever required.

Coming to motivation, StreamER should support any algorithm, metric or reward mechanism. This is because as part of this thesis, there are only a few metrics, and algorithms that can be developed. Anyone who finds the requirement for an evaluation framework must be able to start with the streamER and adapt it to suit their needs. It is impossible at this point to judge the requirements of other people who find the use for streamER. A better way to address this is by designing streamER to be "flexible", to
allow adding and removing of algorithms and metrics. Flexibility should also be extended to the configuration of streamER. Ease of use/development can be counted as an intrinsic design goal as it should be easy for using and developing with streamER. Finally, anyone interested in the framework should be able to run it on their choice of operating system and hardware, leading to the requirement of portability.

4.2 Design Overview

Designing streamER to satisfy design goals set specified in the above section requires a modular design. With the modular design, each component can be made self-contained. Also, each module can then be individually modified without affecting the entire system. Modularization also suits perfectly for streaming data, as each module can decide what data to store and when to store. It also allows ease of development because one needs to understand only the module they are interested in, the rest of the system can be a black box. As an alternative, monolithic designs could be used. Though it might be easy to use in the short term, it will be hard to maintain in the long term. Achieving flexibility in a monolithic design is hard.

Modular design can be easily achieved with object-oriented programming (OOP) development, where each module can be an independent class. Each of the modules can have base properties that can be easily defined in OOP. To implement all these concepts and satisfy portability, a universal platform with a low learning curve is required. Python is one such language which is supported across all major platforms and seems like a major choice from the study of current research during the literature review. There are other languages like Java that can be both universal and used by some researchers. But eco-system of tools and hardware support for python outstrips Java. For example, running a neural network on a Graphics Processing Unit (GPU) using python libraries is much more common as seen in literature study. Due to these reasons, Python is chosen as a language for implementation. Finally, a well-defined communication protocol is required for data and control information transfer between modules.
StreamER: Evaluation Framework For Streaming Recommender Systems

Figure 4.1: Design Overview of the framework

With design principles and concepts are established, the design of StreamER is the next step. The functionality of StreamER is to handle streaming data, provide algorithms with data, receive recommendation data back from algorithms. From the recommendation data, provide rewards and calculate metrics. From this, a basic design based on functionality can be arrived at, where algorithms and evaluator form two different modules. Apart from this, there is a communication interface, configuration and input data that can be counted as modules. A modular design of StreamER evolved from the conclusions above is presented in Figure 4.1. StreamER is comprised of four main modules:

- **Evaluator**: Evaluator is the main part of StreamER, and it provides core functionality for evaluating algorithms. The core functionality includes recommendation handling, reward generation, and metric calculation. Event streaming part of the framework supplies algorithms with event data.

- **Algorithms**: Algorithms under evaluation are part of this module.
StreamER: Evaluation Framework For Streaming Recommender Systems

- **Data**: Data module consists of configuration data and event data required for both evaluator and algorithms.

- **Communication Interface**: Communication between evaluator and algorithms is handled by this module.

### 4.2.1 Evaluation Process

Before progressing further in design, it is important to understand the recommendation process. In a real-world e-commerce site typical user start with a specific item of interest or by clicking some random item shown. Once started it is the job of recommender systems to recommend items to capture the attention of the user. The process starts with first search/browse followed by a series of recommendations from recommender and actions from the user. To simulate the similar environment in streamER, it should capture a dynamic identical to the online interaction between user and recommender algorithm.

![Evaluation Framework Process flow chart](image-url)

Figure 4.2: Evaluation Framework Process flow chart
StreamER: Evaluation Framework For Streaming Recommender Systems

Figure 4.2 presents an overview of the evaluation process that mimics the interaction explained above. Configuration file presented in the chart contains variables determining the behavior of StreamER. The event data file is a sequence of events sorted with a monotonically increasing timestamp, which is the input streaming data.

At the start of the process, the first event in the event data file is read by the evaluator and sent to algorithms followed by a request for recommendations. The evaluator waits until all the algorithms have provided recommendations. Once recommendations are available, the evaluator provides rewards to algorithms for their successful recommendations and calculates metrics. If an algorithm fails to provide recommendations, then its metrics and rewards will end up being a zero. Finally, the evaluator checks for the next event and sends it to the algorithms. The process repeats until all the events are exhausted. It is important to note that the evaluator sends one event at a time to algorithms.

4.3 Data

StreamER operates on configuration, and event data. Following sections provide the detailed description of the same.

4.3.1 Configuration File

Nowadays there are many kinds of recommender systems available, for example, e-commerce, streaming media etc. Each of them has different events of interest (example click, buy, add to cart in e-commerce) and requirements. Conveying this information to the StreamER can be achieved through configuration data file. Configuration file determines the behavior of evaluator, algorithms and the entire system. Table 4.1 presents and describes each individual field of the file. The fields in the table are mandatory and file format is a python standard configuration file.

The framework allows users to modify the configuration file as per their requirement. Event types that trigger the rewards and recommendations are
StreamER: Evaluation Framework For Streaming Recommender Systems

user-defined. Other fields can also be changed and extended in the configuration and are determined by dataset and algorithms.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>DataType</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGORITHMS</td>
<td>Names of the algorithms being evaluated</td>
<td>string.</td>
</tr>
<tr>
<td>HITSET</td>
<td>Types of event data that triggers rewards</td>
<td>integer</td>
</tr>
<tr>
<td>RECSET</td>
<td>Types of event data that triggers recommendations</td>
<td>integer</td>
</tr>
<tr>
<td>METRICS</td>
<td>Names of metrics to be calculated</td>
<td>string</td>
</tr>
<tr>
<td>RECSIZE</td>
<td>Number of recommendations returned by algorithm for each request</td>
<td>integer</td>
</tr>
<tr>
<td>TRAINING_SET</td>
<td>Percentage of initial dataset data for which recommendations are not requested (for future use)</td>
<td>integer</td>
</tr>
</tbody>
</table>

Table 4.1: Configuration File Fields

4.3.2 Event Data File

The event data file is another instance where each of recommender systems can have their own format. It is not practical to extend StreamER to each kind of data file formats. In order to achieve a simple and yet flexible to use the system, a data file format is defined. Input data for streamER must be presented in this format. Organization format of the event data file is \(<session_id, timestamp, item_id, event_type>\). Data is expected to be organized in the monotonically increasing timestamp. The evaluator will not work without a valid event data file, thus making it important to convert the dataset into an event data file format. Table 4.2 describes each of the individual fields.
**StreamER: Evaluation Framework For Streaming Recommender Systems**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionId</td>
<td>Id of the session this event belongs to</td>
<td>It is an integer that can uniquely identify the session</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Timestamp of the event</td>
<td>YYYYMMDDHHMMSS.MS (Y-Year, M-Month,D-Day, H-Hour,M-Minute, S-Second, MS-Milliseconds)</td>
</tr>
<tr>
<td>ItemId</td>
<td>Id of the item</td>
<td>Unique integer identifying the item</td>
</tr>
<tr>
<td>EventType</td>
<td>Type of event eg: 1 (Buy event)</td>
<td>Unique integers identifying various event types matching the configuration file event types</td>
</tr>
</tbody>
</table>

Table 4.2: Event Data File Fields

### 4.4 Evaluator

The evaluator is the heart of the framework, and also complex part of StreamER. To make the design elegant, the evaluator is again subdivided into sub-modules based on functionality.

- **Data Handling:** This sub-module of Evaluator is responsible for reading configuration and event data files. Sending and receiving data to and from the algorithms. As part of the implementation, the evaluator also maintains a database of all recommendations received so far.

- **Metric Generation:** This sub-module is responsible for generating metrics specified in the configuration file.

- **Reward Generation:** This sub-module generates rewards for algorithms for the successful recommendations produced. The success of recommendations is determined from specific event type(s) that trigger rewards, as specified in the configuration file.
4.4.1 Metric generator

Metric generator handles the generation of metrics for the incoming recommendations. In this thesis, metrics are separated from rewards to allow algorithms to test different properties of recommendations. For example, an algorithm might want purchases as rewards, but still, want to measure its recommendation diversity. In the aforementioned example, the metric generator handles diversity as a metric while the reward module provides purchase rewards.

The current state-of-art research contains many metrics developed and categorized. Implementing all of them in the timespan of this thesis is impossible. As part of this thesis, only accuracy metrics are implemented. List of implemented metrics is shown in Table 4.3.
### StreamER: Evaluation Framework For Streaming Recommender Systems

<table>
<thead>
<tr>
<th>Metric</th>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Precision**   | \[ \frac{1}{|S|} \sum_{s \in S} \frac{1}{|L_s|} \sum_{L \in L_s} \frac{|Rel_L|}{N} \] \(4.1\) | Precision is the ratio of the relevant recommendations in lists to all given recommendations aggregated over all sessions.  
Notation:  
L - recommendation list  
s - a session  
S - set of sessions  
N - number of items in the list  
Rel_L - set of relevant recommendations in list L  
L_s - all given recommendation lists |
| **Click Through Rate** | \[ \frac{1}{|S|} \sum_{s \in S} \frac{|L_s^{\text{click}}|}{|L_s|} \] \(4.2\) | CTR is the ratio of relevant recommendation lists that resulted in a click to all given recommendation lists aggregated over all sessions.  
Notation:  
L - recommendation list  
s - a session  
S - set of sessions  
L_s^{\text{click}} - clicked recommendation lists  
L_s - all given recommendation lists |
| **Recall**      | \[ \frac{1}{|S|} \sum_{s \in S} \frac{1}{|L_s|} \sum_{L \in L_s} \frac{|Rel_L|}{|Pur_s|} \] \(4.3\) | Recall is the ratio of relevant recommendations in lists to all the items purchased aggregated over all sessions.  
Notation:  
L - recommendation list  
s - a session  
S - set of sessions  
Rel_L - set of relevant recommendations in list L  
Pur_s - set of items purchased in session s  
L_s - all given recommendation lists |

Table 4.3: List of Metrics
4.4.2 Reward Module

The evaluator is expected to provide rewards when the recommendations from algorithms are successful. Reward generated for a recommendation can be due to events such as the click, adding to cart or purchase of the item for example. Reward module handles the generation of rewards for each of the recommendations from algorithms. Configuration file lists events types for which rewards are to be provided. Rewards can be categorized into two types namely: immediate and delayed.

For data event $E_{i+1}$ at instance $i+1$, a reward $Rw_{i+1}$ is said to be immediate if item $I_{i+1}$ of the event matches the latest recommendation list $R_i$ from the same session as described in Equation 4.4.

$$Rw_{i+1} := \begin{cases} 1 & \text{if } I_{i+1} \in R_i \text{ for given SessionID} \\ 0 & \text{if } I_{i+1} \notin R_i \text{ for given SessionID} \end{cases} \quad (4.4)$$

For data event $E_{i+1}$ at instance $i+1$, a reward $Rw_{i+1}$ is said to be delayed if the item id $I_{i+1}$ of the event matches any of the cumulative recommendations so far $Rc_i$ in the same session as described in Equation 4.5.

$$Rw_{i+1} := \begin{cases} 1 & \text{if } I_{i+1} \in Rc_i \text{ for given SessionID} \\ 0 & \text{if } I_{i+1} \notin Rc_i \text{ for given SessionID} \end{cases} \quad (4.5)$$

Reward Attribution

Attribution of reward is important as it specifies recommendation that triggered the reward. In this thesis, it is decided to simulate the real world conditions where reward attribution has to be implemented inside the algorithm. This is because in real-world an action that can generate reward is not caused by a single recommendation instance. For example, a user can decide to buy a movie when he sees the recommendation for the Nth time where N is greater than or equal to 1. In this sense the attribution is a distribution curve with all the recommendation instances contributing at varying levels.
4.5 Algorithms

Algorithms for evaluation are placed in this module. In the Section 2.2, the hypothesis was made to support the evaluation. According to the hypothesis, this thesis will select two algorithms that provide different and expected output for a given input. Algorithms considered for implementation are judged against that condition. Two simple algorithms that satisfy requirements are the random algorithm and popular-seller recommender algorithm. The random algorithm will provide random items as recommendations from the given list of items. The number of recommendations algorithms are supposed to provide for each event data received is determined from the configuration file. The popular-seller recommender algorithm provides top purchased items so far as recommendations. In this algorithm, the first recommendation will not happen until a specific event is encountered by the algorithm for example buys.

For the given item database, the random algorithm is expected to perform worse than the popular-seller algorithm. This will be true for three metrics implemented, CTR, recall, and precision. The popular-seller algorithm will perform on the same level or better with more number of events processed since popularity will drift through time. The difference in performance is due to the fact that random recommender algorithm recommends items randomly from the item database. The popular-seller algorithm only recommends popular items among the ones bought up to that instance of time. This algorithm also keeps updating based on popularity trend leading to relevant and successful recommendations.

4.6 Communication Protocol and Event Streaming

Communication between the evaluator and the algorithms is designed to be implemented via a client-server architecture where each of the algorithms is implemented as servers providing services to the evaluator. Four kinds
StreamER: Evaluation Framework For Streaming Recommender Systems

of communication messages are transferred between the evaluator and algorithms as shown in Figure 4.3.

<table>
<thead>
<tr>
<th>Session Id</th>
<th>Timestamp</th>
<th>Item Id</th>
<th>Event Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Event Message</td>
</tr>
<tr>
<td>Session Id</td>
<td>Timestamp</td>
<td>Item Id</td>
<td>Event Type</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Query Message</td>
</tr>
<tr>
<td>Session Id</td>
<td>Timestamp</td>
<td>Item Id</td>
<td></td>
</tr>
<tr>
<td>Session Id</td>
<td>Timestamp</td>
<td>Item Id</td>
<td></td>
</tr>
<tr>
<td>Session Id</td>
<td>Timestamp</td>
<td>Item Id</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Recommendation Message</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reward Message</td>
</tr>
</tbody>
</table>

Figure 4.3: Types of Messages

The event message is when the evaluator sends an event from event data file to algorithms. Event messages can contain events like, click, buy, add to cart etc. The query message is when the evaluator requests recommendation from algorithms. The recommendation message is when the algorithm responds with a set of recommendations and finally reward message is sent from the evaluator to algorithms on their successful recommendations. Due to time limitations, this thesis did not implement the client-server model. Instead, a traditional class object functions are used. This kind of implementation does not affect the functionality of the evaluator in any way.

All events mentioned above are sent one after another according to their timestamp and response for each streamed event is processed and this loop repeats. This streaming mechanism forms the backbone for handling the streaming data.
Chapter 5

Implementation

StreamER is implemented based on the decisions made in the design phase. Implementation is carried out in three different phases. The first phase deals with the handling of provided input data. The second phase deals with implementation of the evaluator and final phase deals with implementation of algorithms.

5.1 Input Data

5.1.1 Configuration File

The first part of the data is the configuration file that determines the behavior of the evaluator. It is a simple text file contains information about which types of events the algorithm and evaluator utilizes. As mentioned earlier this file is user-defined and must be finalized before starting the StreamER, and must not be modified during runtime. Configuration file used in this thesis is shown below.

```
[CONFIG]
ALGORITHMS: Random, Popular
#HITSET: 1-Clicks, 2-Cart, 3-Buy
HITSET: 3
#RECSET: 1-Clicks, 2-Cart, 3-Buy
```
5.1.2 Input Dataset

It is important for implementation to showcase various features of the StreamER, a major component contributing to that is the selected dataset. Yoochoose data is one such dataset that can contribute to the validation and evaluation of algorithms, metrics, and rewards. Yoochoose dataset was part of ACM RecSys Challenge 2015[5]. RecSys challenge workshop was conducted in Vienna, Austria in 2015 where the challenge was to predict the probability of a user buying a particular product, given a sequence of click events data in an e-commerce environment. As the dataset is taken from the real world, and from e-commerce where the item and user data is diverse, it is perfect for this thesis.

Dataset comprises of three different files, one file providing buy data, one file providing the click data, and final test file providing only clicks used as test data in the challenge. This thesis only uses click data and buy data files during implementation. Buy file comprises of fields [Session ID, Timestamp, Item ID, Price, Quantity], and click file has [Session ID, Timestamp, Item ID, Category]. Category field of each item in the click file identifies the brand, item category, and special offers such as a discount. Both files are sorted in increasing value of the timestamp.

5.1.3 Preprocessing

The yoochoose dataset is run through different steps of preprocessing. The first step constitutes of removing unwanted fields in files, add an event type field describing click/buy, and finally merging selected files into a single file.

Buy file in the Yoochoose dataset contains fields such as price, quantity which is not important in this context. Click file contains a category field that is also not important. A script was written in python to remove these fields and add event field with appropriate data. The script also merges both files into a single file sorted with increasing timestamp values. This output file is called Event data file. This step is required and performed only once for the entire dataset.

Combined Yoochoose dataset generated in the above step comprises of 37 million events, 52739 unique items, and 9249729 total sessions, thus demonstrating hugeness of the dataset. Working with this size on a normal pc is quite impossible as it takes days to run the entire dataset. Due to this fact, only a subset of first 100000 events from the file is selected. The selected size of data is good enough to showcase StreamER while big enough to capture properties of the complete dataset. This subset of dataset consists of 9284 unique items, and 25197 sessions.

In a typical e-commerce scenario, there will be many overlapping sessions and some of them do not end up with purchases. Session data that does not end in purchases is not relevant in the context of the StreamER. Recommendations given in such sessions cannot be evaluated because only purchase events are used to trigger rewards. A final step of preprocessing is undertaken to remove sessions not resulting in purchases and to generate an item database. Item database consists of a list of all the unique items in the event data file, this is required for algorithms to make recommendations. Though sessions not resulting in purchases are not important for evaluation but, the items referred to in those sessions are important for building item database. During the initialization of algorithms, this item database is sent to the algorithms from the evaluator. Hence this step is run as part of StreamER data initialization.

### 5.1.4 Extensions

Apart from Yoochoose dataset, another dataset was considered and analyzed for implementation. The StreamER was expected to run with this dataset
StreamER: Evaluation Framework For Streaming Recommender Systems
also as an extension of the thesis if time permitted. Unfortunately, time was not sufficient to use this dataset. Though it is not implemented, analyzing the second dataset from a different field identifies shortcomings of StreamER if any. The second dataset considered is the lastfm\textsuperscript{6} data set. It is a music recommendation dataset that would have demonstrated StreamER capabilities in streaming media domain.

Lastfm dataset also contains two data files. The first file contains information about 1000 unique user profile. The second file represents the listening habits of the user with unique track id’s, track names and timestamps. These two files can be merged to form the event data file in the format [UserId, Date, TrackId, Event]. The event data file need to be sorted using the timestamp. Event type for producing recommendations can be a click. Click-through rate metric can be used to measure the accuracy. For streaming applications like lastfm, clicking on any song signifies interest in it and thereby pointing to an accurate recommendation.

5.2 Evaluator

The evaluator is the most significant component in StreamER. As explained in the design phase, it contains metric and reward generator for generation of metrics and rewards respectively. The evaluator is implemented using OOP principles to make it flexible. The evaluator can be extended to include more sub-module other than metric generator and reward module. The evaluator implementation consists of the metric module, reward generator, and output generation.

Throughout the evaluation process, the evaluator has to keep track of all the event data processed so far along with a list of recommendations by each algorithm. Storage of this data allows the evaluator to keep track of events for providing rewards. However storing of data forever poses a problem that streaming data, in theory, can run forever, thereby limiting the amount of data that can be stored. One mechanism to get around this is to define a

\textsuperscript{6}http://www.dtic.upf.edu/ ocelma/MusicRecommendationDataset/lastfm-1K.html
StreamER: Evaluation Framework For Streaming Recommender Systems

memory window, this will be a sliding window in time. Any events falling beyond this window can be discarded. Though this feature is analyzed, it did not make a part in the final implementation.

Evaluation process starts with reading the configuration file, this decides algorithms and metrics to use and create objects for them. Once algorithms, metrics and their properties are determined then initialization functions of implemented metrics and algorithms are called. After this evaluation process described in section 4.2.1 is followed. Individual module functions and their functionality are described in the following sections.

5.2.1 Metric Module

As explained in the design chapter, this thesis implements CTR, precision, and recall as part of StreamER. A generic abstract base class "MetricBase" is defined for the metric module, this class contains functions "InitMetric" and "UpdateMetric". Each of the metrics implemented extends the base class and implement these functions. Table 5.1 explains the class functions in detail.

<table>
<thead>
<tr>
<th>Functions</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitMetric</td>
<td>Number of Algorithms, write to file</td>
<td>Number of algorithms informs how many of them are involved when calculating metrics. Write to file is an optional debug parameter that enables writing internal data structures to a file after each event</td>
</tr>
<tr>
<td>UpdateMetric</td>
<td>Recommendation list, Current Event</td>
<td>Recommendation list contains recommendations from all the algorithms involved, current event is the latest from the event data file</td>
</tr>
</tbody>
</table>

Table 5.1: Metric Base Class Functions

For calculating CTR, recall, and precision two classes "MetricCTR" and
StreamER: Evaluation Framework For Streaming Recommender Systems

"MetricPrecision" respectively are implemented in their individual files. The recall is also implemented using the same class since both recall and precision complement each other. For calculating CTR, clicks and recommendation count are kept track by the metric module. For calculating precision and recall, purchases and recommendations count are stored. The evaluator considers only recommendations received for current event data. Update metric function checks if event type matches expected value (for example: Buy for precision and click for click-through rate) and checks if event data matches at least one of recommendations. If this check is successful, the corresponding click, or buy count is incremented. Call to UpdateMetric function results in the increase of recommendations aggregate. Finally, metrics are calculated using the formulas provided in the Table 4.3. Also, there is a debug feature where each time UpdateMetric is called, internal data structures and function parameters are written to a file. It is important to note that recommendations from individual algorithms can be identified via their algorithm ID. This will enable the metric generator to keep track of successful recommendations of each algorithm and calculate metrics individually for each algorithm.

5.2.2 Reward Generator

For each tuple of event data, and recommendations from the algorithms "Reward_Calculator" function is called by the evaluator. This function is implemented in the evaluator. Event data and recommendations are passed as arguments for this function. This thesis implements delayed rewards, i.e, all the recommendations up to that point for that session are matched for when calculating reward. Reward module compares if the event field matches configuration option that describes event type for which reward is provided. If it is successful, then a reward is provided to all the algorithms that contained the winning recommendation. As of this final implementation, rewards are only provided for successful recommendations based on purchases.
5.2.3 Output Generation

Output in this thesis is generated via graphs in real-time. Python library called Matplotlib is used to plot the results generated by metric and reward modules. Each graph contained the comparison of metric results for both the algorithms used in this thesis. There is also a debug feature which allows output data and internal data to be written to files with predetermined names.

5.3 Algorithms

Algorithms have a passive role in this framework. The work-flow of algorithms is shown in the Figure 5.1. After initialization, they wait for event data from the evaluator. Upon receiving event data, algorithms process data and produce recommendations. Algorithms respond in a sequential manner to the requests and generation of recommendations completely depends on the underlying mechanism used in the algorithm. As part of StreamER, random recommender algorithm and popular-seller recommender algorithms are used. Former uses the randomization technique to recommend the random items while the later recommends items that users are buying frequently.
In order to make the recommendation process more effective algorithm need to be trained often. To facilitate the training process algorithm is provided with access to all the event data from session up to the present time. It also has access to the item database that is a list of items in the data set with item ids and metadata (if available). This helps the algorithm to take a look back at the events from previous sessions or from the current session and make the recommendations in the given session based on previous knowledge which will lead to more accurate predictions.

An algorithm base class is implemented using an object-oriented mechanism to provide room for extension. The users of this evaluation framework who want to test their algorithm with the evaluator must extend the algorithm base class and implement three functions provided by it as shown in Table 5.2.

Algorithm waits until recommendation request is received from the evaluator. If no request is received then it waits. For every received request algorithm need to give a response in the form of recommendations. If algorithms do not respond to recommendation request evaluator need to wait since the recommendation request is blocking. In case if the algorithm is
unable to produce recommendations at that time, it will return an empty set of recommendations to unblock the evaluator. As a response to recommendations, it receives a reward for all the successful predictions it has made. Recommendations are considered successful only if the recommended item is bought by the user within the same session where the recommendation is made. However, as specified earlier the reward triggers can be configured as per requirement. The Algorithms are also expected to produce recommendations as long as they are receiving recommendation request from the evaluator.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Handler</td>
<td>Handles the event data from evaluator</td>
</tr>
<tr>
<td>RequestRecommendation</td>
<td>Provides recommendation for evaluator</td>
</tr>
<tr>
<td>RewardHandler</td>
<td>Handles rewards</td>
</tr>
</tbody>
</table>

Table 5.2: Abstract functions of Algorithm

5.4 Communication Protocol

The aim of communication protocol is to facilitate communication between different modules of StreamER. Idea is to implement it using client-server communication mechanism. Apart from the other ideas such as event notification techniques were analyzed for use. One such event library considered was Pypubsub\cite{pypubsub}.

Pypubsub is an asynchronous event library that provides a publisher-subscriber event framework. In pysubhub, modules that provide events are registered as publishers and modules that wait for various events are registered as subscribers. Registering subscribers also includes a callback function that will be called whenever a message is posted on that event. There can be more than one subscriber for each given event. When a publisher posts

\cite[http://pypubsub.readthedocs.io/en/stable/]
message on an event, it will be blocked until all subscribers have received the
message. Acknowledgment for received messages is achieved in this way.

As explained above, pypubsub is an excellent framework to use when there
is one-way asynchronous communication is required between one publisher
and multiple subscribers. In this thesis, evaluator and algorithms have two-
way communication as explained in the Section 4.2. Pypubsub does not
contain support for multiple publishers and single subscriber on the same
event, rendering the framework useless for use in StreamER.

It is clear from the analysis that a client-server model is optimal for
StreamER. But this conclusion has arrived quite late in the implementation
phase. Therefore, simple blocking function calls with arguments are used in
this thesis for communicating between different modules.
Chapter 6

Evaluation and Results

6.1 Results

Implemented StreamER is run on the selected subset of yoochoose dataset. Results of the calculated metrics are plotted in graphs. In the Figure 6.1 precision is plotted against the number of queries each of the algorithms has received. It can be seen that popular-seller algorithm starts at zero and sees a transient jump after which it stabilizes. A manual inspection of the input data and debug data reveals few continuously successful recommendations by the algorithm. The sharp decline at around 3000 queries is due to a streak of unsuccessful recommendations. After this valley precision slowly attains an equilibrium. When it comes to precision for random algorithm recommendations, it is almost close to zero. Manual inspection of input and output data shows there are very few successful recommendations, eg: below 20 in one run, thereby resulting in close to zero ratio. Results satisfy hypothesis that popular-seller recommender performs better than random recommender algorithm.

It is important to note that the graph for precision also identical to the graph for rewards. This is because the formula for rewards ratio and precision is identical since they both are calculated for purchases against recommendation lists.
In the Figure 6.1 click-through rate is plotted against the number of queries each of the algorithms has received. It can be seen that Popular-seller algorithm starts at zero and sees a transient jump after which it stabilizes. A manual inspection of the input data and output data reveals a few continuously successful recommendations by the algorithm. The decline, at around 1500 queries is due to a streak of not successful recommendations. After this valley click-through rate shows decline up to 4000 items after which it slowly attains an equilibrium. The slow decline is due to fewer clicks being correct. When it comes to CTR for random algorithm recommendations, abysmal success can be seen from the graph. It is also important to note that random recommender algorithm graph changes every time the evaluator is run.
StreamER: Evaluation Framework For Streaming Recommender Systems

Figure 6.2: Click Through Rate

Figure 6.3: Recall
Finally, Recall plot for algorithms can be seen in Figure 6.3. It follows the same pattern as the other two metrics with popular algorithm performing better than the random algorithm. This also means the same reasons have contributed to the shape of the graph.

6.2 Evaluation Discussion

Results presented in the previous section have clearly demonstrated that the StreamER is working as expected. As presented in the research section, this thesis is expected to be evaluated for both functionality and accuracy. Evaluation for functionality can be answered by analysis of each of individual modules.

Starting with the data module, The first part is the configuration file. As presented in the implementation section evaluator have read the data file and used it for configuration. One way to test this is to change various fields in the configuration file and see if it affects the behavior of the evaluator. As part of the evaluation, all fields were changed to see corresponding changes in the behavior of the evaluator. It also has resulted in finding few limitations of current implementation explained in the next section. Also changing the length of the input data has resulted in longer evaluator runs and at one point when 10 million data items were run, the evaluator crashed after 2 days. This once again has identified limitation of processing power and memory management.

The functional correctness of evaluator can be easily proven due to the fact that, data is passed between modules from event data file, metrics and rewards are being calculated. In short, the accuracy of results also proves functional correctness. To strengthen the argument, manual verification of debug files show data from each of modules being written for each iteration. Same arguments can be applied to the evaluation of the functional correctness of algorithms. As graphs metric calculators are registering accurate output from algorithms, this means they are functioning correctly. In the end, the functional correctness of the StreamER is evaluated ipso facto from the accuracy of results and manual inspection.
StreamER: Evaluation Framework For Streaming Recommender Systems

Evaluation for the accuracy of the StreamER is proven by results produced in the above section, and the fact that they conform to the hypothesis produced. Also as a double validity test, data in random steps and wherever change in trend happened is manually inspected. This also helped to explain the shape of output graphs. During the manual inspection, each of recommendations from popular recommender item is cross verified at some places to see if it matches the most popular items so far. Places of interest for manual verification were the places that resulted in the change of popularity, along with few items in the vicinity. Random algorithm output is easy to verify due to the fact it kept producing random items at each turn and the results were different always. Metrics are then verified manually by comparing recommendations, event data and the value of the metric calculated. Finally, rewards were also verified manually in the same way.

![Reward Ratio for Algorithms](image)

Figure 6.4: Rewards

Rewards ratio presented in Figure 6.4 is calculated as a ratio of aggregate rewards to aggregate recommendations. The reward curve is very important for one reason, it was calculated on a different run than the graphs shown.
StreamER: Evaluation Framework For Streaming Recommender Systems

above. In this thesis, precision is calculated for buys thereby effectively making it the exact formula used for calculating immediate rewards. So Figure 6.4 matching Figure 6.1 for popular items not only means output is constant for a given data at any given run of StreamER but also the accuracy of reward and precision calculation. This thesis also identifies the possibility of them both being wrong, but manual inspection step has removed that possibility. The difference in random algorithm output shape from two figures is proof that random algorithm behavior indeed varies between each run of the StreamER.

From this discussion, it can be concluded that StreamER is successfully evaluated for both functionality and accuracy.

6.3 Limitations

The final product of this thesis, StreamER has some limitations in its current implementation due to the limiting time factor. These limitations are encountered in different phases, such as analysis, implementation and finally in the evaluation. Each of the limitation and a possible solution are discussed below.

The first and foremost challenge encountered was during the analysis phase and it is with data size. PC this thesis was developed on had limited processing power as it was running totally on a 3rd generation Intel Corei7 CPU with no graphics card to offload calculations. This resulted in the exponential increase of time to complete the simulation with an increase in the data set. To get around this limitation, a limited data set was used.

Another limitation with data size is encountered during the implementation phase. As the evaluator was storing event, and recommendation data from the start of the run, it was running out of memory after some time. This also has a side effect of slowing down the simulation due to unavailability of the RAM. To remedy this data after a certain time should be discarded, this will limit the data usage and also allows running even bigger datasets.

Communication protocol implementation is another limitation from the implementation phase. The client-server model described to establish the
communication between the evaluator and the algorithm was not used. This has a limited system by requiring direct communication. Implementation of this is simply a matter of time.

Directly following the previous limitation is multi-threading of the prototype. If the client-server model is implemented, then multi threading allows parallelization of StreamER execution. This can result in increased speeds of StreamER and better machine resource utilization. This also can allow more algorithms, and metrics to run in parallel.

During implementation, only the yoochoose dataset is used. As explained in section 5.1 analysis of the lastfm dataset is conducted but not used. This once again was limited by time and easy to implement.

Finally, during evaluation, it was observed that the evaluator was using hard-coded event types for calculation of metrics. This is because configuration file does not have a field to specify the same. This is because each metric can use a different event type for calculation, for example, CTR uses clicks whereas precision uses buy. To solve this limitation, the configuration file has to be extended to allow each algorithm metric to specify event types of their choice.
Chapter 7

Conclusions and Future Work

7.1 Conclusion

The goal set at the beginning of the thesis to design and implement an evaluation framework StreamER is successfully achieved. Proof of concept is provided by implementing the prototype in accordance with the design proposed. Results and discussion provided in the previous chapter allow concluding that the framework is implemented and evaluated successfully. Though the artifact implementation is successful, it is important to observe if the research goal was met and the research questions were answered.

The first sub-research question ”How to design the evaluator that is flexible, and easy to use?” was answered in multiple stages. Configuration file provides part of the flexibility, choice of OOP paradigm gives ease of use and flexibility and modularization provides the final piece. Answering the question, the Evaluator can be designed for flexibility and ease of use by providing options for configuration, choice of proper implementation paradigms such as OOP and modular design.

The second sub-research question ”How to design a communication protocol between the evaluator and the recommender system algorithm in such a system?” was answered in the design analysis and implementation phases. As part of the design analysis of StreamER, requirements for communica-

StreamER: Evaluation Framework For Streaming Recommender Systems

A communication protocol were identified. One requirement is two-way communication channels between modules along with asynchronous data transfer mechanisms. Design of such protocol must also support blocking and non-blocking requests. So any communication protocol satisfying these conditions works for StreamER, and once such protocol is the client-server model. Though not implemented, design requirements gathered thus answering the second sub-research question.

Answering both sub-research questions also provides a partial answer to the research question, ”How to build an off-line evaluation framework for streaming recommender systems?”. Rest of the answer lies in the fact that StreamER runs on serial data, thereby processing part of the stream at any given time. Every part of StreamER assumes input data as a stream and does not expect the presence of complete dataset. Design decisions, when combined with the Object-oriented approach, provides an answer to the research question.

Finally ”Design and Implementation of an Evaluation Framework for streaming recommender Systems” is demonstrated via the results and answering research questions.

7.2 Future Work

Though some of the work is done as part of the thesis, there are concepts and ideas that can make StreamER better. Algorithms implemented as part of the thesis were simple random recommender and popular-seller. It would be interesting to test with more complex algorithms in the future.

As part of the thesis, only three metrics were calculated, CTR, Precision, and Recall. All these metrics are accuracy metrics and expanding them to include diversity metrics is important.

Since the aim is to evaluate streaming recommendations, the addition of the timing component to measure the algorithm running time would be interesting.

It might be interesting for researchers collaborating in the same department to use StreamER in local area network. If the client-server model is
StreamER: Evaluation Framework For Streaming Recommender Systems

implemented, then the sockets can be used to accept connections from algorithms running on computers on the network. An extension of above would be also adding support for online evaluation.
StreamER: Evaluation Framework For Streaming Recommender Systems

Bibliography


[6] Michael D Ekstrand, Michael Ludwig, Joseph A Konstan, and John T Riedl. Rethinking the recommender research ecosystem: Reproducibil-
StreamER: Evaluation Framework For Streaming Recommender Systems


[12] Emanuel Lacic, Matthias Traub, Dominik Kowald, and Elisabeth Lex. Scar: Towards a real-time recommender framework following the microservices architecture. 2015.

StreamER: Evaluation Framework For Streaming Recommender Systems


[22] Zaigham Faraz Siddiqui, Eleftherios Tiakas, Panagiotis Symeonidis, Myra Spiliopoulou, and Yannis Manolopoulos. xstreams: Recommending items to users with time-evolving preferences. In Proceedings of
StreamER: Evaluation Framework For Streaming Recommender Systems

*the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14)*, WIMS ’14, pages 22:1–22:12, New York, NY, USA, 2014. ACM.

