Stream-Based Recommender Systems

Internship Report (2 months)
Sep 7, 2020

Author:
Dina El Zein
École Normale Supérieure de Lyon

Supervisor:
Marie Al-Ghossein
LTCI, Telecom Paris

Abstract

Recommender systems (RS) are tools that aim to recommend relevant items for users. The main challenge regarding the use of batch recommender systems is real time analytics data; the model has to be retrained periodically from scratch, which becomes costly over time. This report introduces a python framework for developing stream-based recommender systems, including the implementation of collaborative filtering algorithms and evaluation metrics, ensuring replicable experiments.

Contents

1 Introduction

This introduction first aims at giving general information about Recommender Systems (RS) and introduces stream-based RS.

1.1 Recommender Systems

RS are tools that aim to suggest products for a user based on his/her preference. In a very general way, RS cover techniques that aim to predict personalized suggestions according to users' preferences and current needs. Recommending content is a very important task in information systems, specially with the rapid growth of available content. For example, online shopping like Amazon gives each customer personalized recommendations of products that a user may be interested in. Other examples are movies, like Netflix recommends movies to customers.

There are three main source of information to consider when working with RS: knowledge of the user, knowledge of the item, and the interaction between users and items. The knowledge of the user can be: demographic data (age, gender, etc.) and historical data, i.e interactions with items, purchases, views, clicks. The knowledge of the item usually covers content information, i.e features, description.

There are two formulations for the recommendation problem: the Rating Prediction (RP) and the Item Prediction (IP). RP predicts the rating that a user might give to an item with which he didn't interact yet, in other words, to predict missing values in a feedback matrix (feedback matrix is a numerical positive ratings matrix). Where as, IP is to predict whether a user will like an item or not.

There are two types for feedback: explicit feedback and implicit feedback. Explicit feedback is a rating given by a user to an item where as implicit feedback is clicks, view of item description, etc. The feedback matrix can be either explicit (positive ratings from users to items) or implicit (can be seen as a boolean value matrix where true value (1) indicates a positive user-item interaction and a false value (0) indicates a negative user-item interaction). There are some Limitations for both types of feedback. For explicit feedback, the limitations can be: users don't provide explicit feedback always, user's influence by social circle (family, friends), and users usually give higher or lower ratings for appreciation or depreciation. Hence explicit feedback might be biased. For implicit feedback, the main challenge is that the data is very noisy, i.e every item the user interacted with is considered as a positive item whereas this is not always the case. Adding to that, the negative feedback is missing; negative items are considered as a mixture of unknown items (for a user) and unliked items (i.e one click for the item).

Batch RS which is similar to classical machine learning (ML) is the most popular functioning method of RS. Models for batch RS are build based on large static dataset, and then rebuild it periodically as new chunks of data arrive and added to the original dataset. In that setting, around 80% of the data is taken to learn the model, then test the trained model on the left 20%. There is no learning in the testing part as the model is static.

Batch RS encounter some limitations: The user feedback is not captured by the model as soon as it is being observed, which results in the generation of recommendations that are not adapted to the current situation on some specific types of data. For example, considering the news domain, news articles are being continuously added, the model should be updated every time new articles are added to the system and users interact with the articles; hence the latest news are taken into consideration [JJK18].

1.2 Stream-Based Recommender Systems

Data stream mining is one way to address the problem of batch RS. It is an online RS that learn from continuous data streams of interactions and provide recommendations in real-time. In that setting, the model can be trained over around 20 % of the data to initialize the model and to avoid the cold start problem (the cold start problem is related to the sparsity of information, i.e when new users or items are added) then testing and training incrementally on the left 80 %.

Stream-based RS implement some algorithms that can immediately take into consideration the latest events into their predictions. These algorithms include core algorithms (based on user-items interaction and contextual information) and some trivial baselines: Recently Published, that recommends the latest items; most popular, the overall most clicked products; recently popular, the items that were clicked the most during the last N minutes [Guo+15].

The goal of our work is to create an incremental version of some recommendation algorithms so that they adapt their recommendations to the latest update in the data (items) during the user's current session and immediately consider new items and new users interaction.

1.3 Framework for Stream-Based Recommender Systems

In this work, we implement a python framework for stream-based RS, with the following goals in mind:

- Implementing a set of recent state of the art recommendation algorithms
- Reproducing results for different RS' algorithms
- Providing a tool facilitating the implementation of recommendation algorithms and comparing them in the same setting
- Evaluating the performance of the methods on public datasets

2 Frameworks for the Development of Recommender Systems

This section compares different existing frameworks for RS.

2.1 Frameworks for Recommender Systems

Here are some open source frameworks for the development of RS:

- *Librec* is an open source java framework that implements a suite of state-of-the-art recommendation algorithms as well as traditional ones. Librec, provides similarity based diversity measures as well as the traditional accuracy-based measures [Guo+15].

Libraries	Type	Language	Algorithms
Librec	Batch	Java	BL and CF algorithms (IP, RP)
LKPY	Batch	Python	Item based KNN, User based KNN
StreamingRec	Stream	Java	BL and CF algorithms
Alpenglow	Stream	C++	ALS and Assymetric Factor, NN, Popularity, SVDPP

Table 1: Characteristics of the recommender systems libraries (Refer to the list of symbols to check the abbreviated words)

- *StreamingRec* is an open source java framework implements some algorithms that can immediately take into consideration the latest events into their predictions as well as short term interest of the user [JJK18].
- *Lenskit* is an open source java framework that provides tools and infrastructure support for managing data and algorithm configurations, implementations of several collaborative filtering algorithms, and an evaluation suite for conducting offline experiments [Eks19].
- *LKPY* implements a python version of Lenskit that supports new algorithms. It's a successor of Len's kit in terms of efficiency, new algorithms, and language [Eks19].
- *Alpenglow* is an open source C++ framework that runs the models in three different methods: batch model, online model, batch and online model [Fri+17].

2.2 Comparison between Frameworks

Table 1 shows a brief comparison between the characteristics of the libraries presented above.

2.3 Our Framework for Stream-Based RS

As seen in section 2.2, there are two frameworks for stream-based RS, which are Alpenglow and StreamingRec. Alpenglow is a C++ framework with python API that implements few batch and online RS models. It produces temporal recommendation models (are models that account for concept drifts and non stationary effects) that can be combined with batch models to achieve significant performance gains; the framework for Alpenglow can simulate the streaming recommendation scenario offline (trained continuously after each observation)[Fri+17]. As for StreamingRec, it is a Java framework that implements a variety of RS models. The main goal of StreamingRec is to address challenges of benchmarking news in a session based strategy [JJK18]. Session based strategy is the task of recommending the next item based on previously recorded user interactions.

There are some reasons for implementing a stream-based RS framework: firstly, Alpenglow is a complex framework in terms of reading and testing as it's written in C++; C++ is a complex language for most developers, it makes it harder for developers to add new methods to an existing framework or even testing it (unit testing); secondly, StreamingRec considers a very specific setting which is the session-based strategy for news domain.

According to Nautiyal [Nau18], python is considered to be in the first place for the list of all development languages; it supports object oriented, functional, as well as procedure oriented styles of programming. Because libraries in python make our tasks easier, we implemented a python framework that implements the most important RS models. The framework works with data offline that performs online recommendation, preceded by a brief phase of training in batch which is different from the methodology used in Alpenglow and StreamingRec.

3 Framework for Stream-Based Recommender Systems

This section discusses the stream-based recommender models we considered in this work, in addition to some evaluation metrics.

3.1 General Functioning

The main goal of our stream-based RS framework is to address the challenges of real time data analysis and take into account new items and user interactions as soon as they are generated.

The python stream-based RS framework consists of several modules that are cited in the following:

- **Data Preparation** which consists of:
  - **Load:** Loading the dataset
  - **Organize:** Sorting the datasets in increasing order based on the timestamp.
  - **Convert:** Converting the the ids of the users/items in the datasets by mapping them to incremental indices (0,1,2, ...).
  - **Divide:** Dividing the dataset into train/test datasets.
- **Recommending and Evaluating Model** consists mainly of:
  - **Fit the model:** using the training data, train recommendation model = learning the model (f) on the training dataset (train). This is done in the training phase.
  - **Predict, update, and evaluate the model:** are done in the testing and evaluating phases as follows:
    For every observation (u, i, t) in the test dataset:
    - **Predict:** use f to generate top recommendation for the active user (u):
      - For every item in the catalog:
        Compute f(u,i) and store it in list "score-recommendations"
      - Sort "score-recommendations" in inverse order

        - Recommendations for u = score-recommendations
    - **Evaluate:** Evaluates the recommendations list against the single relevant item that the user interacted with $(u, i,$ score-recommendations$)$:
        - evaluate metric$(u, i,$ score-recommendations$)$ = compute metric$(u, i,$ score-recommendations$)$
        - Store the value of metric = append value to metric values[ ]
    - **Update:** updates the model on the revealed user-item interaction (u, i)

3.2 Recommendation Models

RS make prediction based on users' historical behaviors. It predicts user preference for a set of items based on past experience. To build a RS, the two most popular approaches are Content-based and Collaborative Filtering (CF).

Content-based filtering: is a technique used by RS to recommend items based on a comparison between the content of the items and a user profile. Content-based filtering approaches require information about the content of items, taking the form of features or unstructured text; for example, in a movie database, the movie features can be genre, year, director, actor etc [Luo18]. These approaches were not considered in this work .

Collaborative Filtering is a technique used by RS that can filter out items that a user might like on the basis of reactions by similar users. On the contrary of content based algorithms, CF algorithms don't need any information about items, but rather rely on user interactions. The main assumption of CF algorithms is that the users who have agreed in the past tend to also agree in the future [Luo18]. CF algorithms are usually built on a rating given by the user to the items. However, in the context of RS for the web (which is our case), ratings information are limited; we only have access to the items the users saw, purchased ... This kind of data is usually saved as binary implicit ratings (1 if the user has seen the item and 0 otherwise.).

3.2.1 Most Popular

This model recommends for all users the most popular items. One can calculate the frequency of each item (i) and recommend the most popular ones for each user (u). A pseudo code for this model can be found in appendix 12.

3.2.2 Neighborhood-Based Approaches

Neighborhood-Based approaches advise the user about what products/items to purchase based on a similarity measure between users or items. There are two main approaches for Nearest Neighborhood (NN): item-based and user-based. The user-based approach proceeds by looking for N users in the dataset (D) that are most similar to u; the items preferred by these neighbors are the recommendations to make. In the item-based approach, we look for N items which are similar in terms of users that preferred them to the item in the session. Hence, item-based algorithms may be able to provide the same quality as the user-based but with less computations due to the fact that the number of users in a dataset is usually greater than the number of items; however, we may always recommend to the user items of similar taste. Because NN algorithms are mainly based on the similarity function which is usually represented as a similarity matrix,

one way to keep the recommendations up to date is to refresh the similarity matrix from a time to time. It would be very expensive if we want to build the matrix every time from scratch; that is why we followed an incremental approach. The authors in [MJ09] have introduced an incremental version of the k-nearest neighborhood item-based approach; a user-based approach was proposed by Papagelis et al [Pap+05].
The similarity measure between items and users can be defined in many ways. In our implementation we used the cosine distance measure. Let n and m be the number of items and users in the database respectively.

User based similarity: let users u and w be two users in the m dimensional users space. The similarity between u and w is measured using the cosine similarity (sim).

$$sim(u, w) = \cos(\vec{u}, \vec{w}) = \#(U \cap W)/\sqrt{U} \times \sqrt{W} \tag{1}$$

where U and W are the sets of items the users u and w evaluated.

Item based similarity: the similarity between items i and j is defined as follows:

$$sim(i, j) = \cos(\vec{i}, \vec{j}) = \#(I \cap J)/\sqrt{I} \times \sqrt{J} \tag{2}$$

where I and J are the sets of users that evaluated items i and j respectively.
A pseudo code for the K nearest neighbor item based $(KNNI)$ approach can be found in appendix 2. A similar user based approach for NN is implemented using the user-based similarity measure.

3.2.3 Matrix Factorization

Matrix Factorization (MF) for CF is inspired by Latent Semantic Indexing, a technique to index large collections of text documents. In a CF problem, the same technique can be used in the user-item rating matrix, uncovering a latent feature space that is common to both users and items. The authors in [VJG14] have proposed an incremental version of the MF algorithm and here how it works. Suppose we have a rating matrix R, the MF algorithm decomposes it into two matrices A and B using the classic Stochastic Singular Value Decomposition (SVD) where matrix A spans the user space and matrix B spans the item space. Given this, we can predict the rating for an item u by the dot product between A and B s.t. $R_{ui} = A_u.B_i$.

Training is performed in order to minimize the error (err):

$$min_{\mathbf{A},\mathbf{B}} \sum_{(u,i) in D} (R_{ui} - A_u.B_i^{\intercal})^2 + \lambda(||A_u||^2 + (||B_i||^2) \tag{3}$$

where D is the set of user item pairs for which the rating is known and λ is the regularization parameter.

Given a training set in the form <user,item, rating> $(< u, i, r >)$ where rating in our case is a binary 1 (rated item) or 0 (unrated item), we construct the feedback matrix R. A false value in the matrix R has two distinct meanings either the user dislikes the item (negative preference) or did not interact with the item (unknown preference). In our current work, we assume that false values are missing values as opposed to negative ratings.

SGD iterates multiple times over the dataset until a stopping criteria is met. At each iteration (user, item), sgd calculates the $err_{ui} = R_{ui} - \widehat{R}_{ui}$ and then updates A_u and B_i accordingly s.t.

$$A_u \leftarrow A_u + \eta(err_{ui}Bi - \lambda Au) \tag{4}$$

$$B_i \leftarrow B_i + \eta(err_{ui}Au - \lambda Bi) \tag{5}$$

where η is the learning rate and λ is the regularization parameter.

In our case, given a < user, item > (<u,i>) interaction, it means that u interacted with i because she likes it. So $R_{ui} = 1$ and $\widehat{R}_{ui}$ which is the predicted error is $A_u B_i^{\intercal}$. Hence the error in our case is $error = 1 - \widehat{R}_{ui} = 1 - A_u B_i^{\intercal}$. Refer to the pseudo code for this algorithm in appendix4.

3.2.4 Matrix Factorization with Negative Feedback (MFNF)

Another version of MF which considers negative feedback is implemented by the authors in [VJG15]. This method has a very similar task to MF but it takes some artificial negative items for every positive item. When there is no explicit rating in the data, RP algorithms are not applicable. Instead, data consists of positive-only user-item interactions (so negative feedback is absent), and the task is therefore not to predict ratings, but rather to predict good items to recommend (IP). Here comes the question about the absent data and how do we deal with it. Do we consider it as negative data (unliked items) or unknown preferences? Although it's not easy to distinguish between them when only positive feedback is available.

The strategy is to create a set of negative items for every positive user-item interaction. So for every user, item (u, i) positive interaction, l active negative items $(u, j_1), \ldots (u, j_l)$ will be taken from the steam of data. The active negative items are chosen from a queue that contains all items seen so far, every time an observation occurs, the item is added to the queue (tail of the queue) and l items are selected from the head of the queue as negative items. A pseudo code for this method can be found in appendix 5.

3.2.5 Matrix Factorization with Negative Feedback BPR (MFBPR)

Another way to process negative feedback is to use Bayesian Personalized Ranking (BPR) for implicit feedback; this strategy is introduced by the authors in [Ren+12]. In this process, they created a dataset D from the training set (S) by giving pairs (u,i) in S a positive class label and all other combinations in (U × I)\S a negative one. We denote that if user u interacted with item i_2 and not with i_1, that u prefers i_2 over i_1 : $i_2 >_u i_1$. For items that have both seen or unseen with the user, we can not refer any preference. The dataset D is formalized as follows: $D_S : U \times I \times I$ s.t.

$$D_S := (u, i, j) | i \in I_u^+ \cap j \in I \backslash I_u^+ \tag{6}$$

where

$$I_u^+ := \{i \in I : (u, i) \in S\} \tag{7}$$

We focus on learning this model using the SGD method (as the MF method presented above), but with a pair wise approach (for each user, we consider a positive item and a negative item) to minimize the error. Let $p_t = ((u, i), (u, j))_t$ denote a pair of training instances sampled at time t, where $(u, i) \in$ S has been observed in the stream and (u, j) $\notin$ S (so j is a negative item). Formally, we define the set P as the set of tuples $p = ((u, i), (u, j))$ selected from the data stream S, as follows:

$$P := \{((u, i), (u, j)) | i \in B_u^+ \cap j \notin B_u^+\} \tag{8}$$

where

$$B_u^+ := \{i \in I | (u, i, x_{ui}) \in S\} \tag{9}$$

With that defined P, we have to find A and B s.t. $R = AB^\intercal$ that minimizes the pairwise objective function:

$$argminL(P, W, H) + 2\|W\|^2 + 2\|H\|^2 \tag{10}$$

There are several techniques to sample negative items:

- **Single Pass**: According to authors in [Dia+12b], for every observation in the stream, single pass chooses a random negative item and updates the model. This approach doesn't remember previously seen data so it doesn't remember B_u^+, that is why the negative item is chosen in a random way.
- **User buffer**: As mentioned by authors in [Dia+12b] a buffer B_u^+ is constructed for each user but in a limited space(b). The buffer keeps track of the most recent items for each user in memory. After each observation (u,i) in the stream, the buffer Bu+ is updated as follows:
  if $|B_u^+| < b$ then $B_u^+ = B_u^+ \cup i$
  else:
  Delete the oldest instance from B_u^+
  $B_u^+ = B_u^+ \cup i$
- **Reservoir Sampling:** As explained by authors in [Dia+12b], reservoir sampling involves retaining a fixed size of observed instances in a reservoir. The reservoir should represents a "sketch" of the data in a fixed size. The reservoir is represented as a list of items: R := $[s_1, s_2..., s_{|R|}]$. After each observation in the stream, the reservoir is updated. We continue adding pairs of data (u,i) into the reservoir until it reaches its maximum size, then we start to update it by selecting a random element from the reservoir to delete, then add the new observation. Notice that instances can occur more than once in the reservoir reflecting the distribution of the observed data. This way we keep track of some of the old data in a limited space. In contrast to the user buffer, we don't limit the space for each user, but instead choose randomly items from the data (stream) and save them into the reservoir and update the model randomly using the reservoir.

As a first step, we decide to implement the third one for two main reasons:

- First, we may not have enough memory to save a buffer for every user, specially that we are working on a streaming data so we may have billions of data reaching at once.
- Second, because the single pass chooses the negative item randomly, then there will be a high probability of choosing the negative item for the user from its positive ones. This way, we don't improve the MF presented above.

According to DA's strategy [Dia+12a], the negatives items for each observation are chosen based on an active learning methodology that chooses the most informative examples present in the reservoir. From the reservoir, a small buffer is constructed for the user (in the interaction) based on the "59 trick" that samples 59 instances from the reservoir (if available), this is enough to guarantee with 95 probability that one of them is among the top 5% closet instances to the hyperplane (that divides the positive and negative classes for the user). For the observation we want to update the model on (u,i), and from the 59 negative items (if

	1M MovieLens	1M MovieLens with ratings = 5
Events	1000209	226310
Users	6040	6014
Items	3706	3232

Table 2: 1M MovieLens Dataset Characteristics

available), one negative item i_j is sampled for a user u with a probability proportional to $1/\delta_{uij}$ s.t δ_{uij} is the distance between the positive item i and the negative item j for the user u. With a very high probability, the negative item that has the smallest δ_{uij} will be chosen. We have tried to implement this method using the pseudo code in appendix 6.

3.3 Evaluation of Recommendations

Recommendation models can be evaluated using different metrics. Some metrics evaluate classical top N recommendations, others give the accuracy of the recommendations.
Three evaluation metrics are implemented:

(i) **Mean Reciprocal Rank (MRR)**: measures the rank of the recommendation list as follows:

$$\frac{1}{rank(item)} \text{in the sorted recommendation's list} \tag{11}$$

(ii) **Recall@N**: measures the percentage of relevance of an item in the recommendation list as follows:

$$Recall@N = 1 \text{if the item "i" is included in the first N recommendation list, 0 otherwise} \tag{12}$$

(iii) **Discounted Cumulative Gain (DCG@N)** measures the quality of the top N recommendations as follows:

$$\frac{1}{log_2(rank(i)+1)} \text{if the item "i" is included in the first N recommendation list, 0 otherwise} \tag{13}$$

4 Experiments

We conducted a number of experiments with the presented implemented algorithms of our framework to assess their performances. These results are shown below.

4.1 Datasets and Experimental Procedure

We report the results of the above recommendation models on the 1M MovieLens dataset consisting of movie ratings [03].

Table 2 shows the description of the 1M MovieLens dataset.

	$K = 5$	$K = 10$	$K = 20$	$K = 50$	$K = 100$
MRR	0.016	0.016	0.016	0.016	0.016
Recall@1	0.006	0.006	0.006	0.006	0.006
Recall@5	0.0145	0.014	0.014	0.014	0.014
Recall@10	0.0247	0.0247	0.0247	0.0247	0.0247
Recall@50	0.08	0.08	0.08	0.08	0.08
Recall@100	0.19	0.19	0.19	0.19	0.19

Table 3: Evaluation results for nearest neighbor, user based model

The 1M MovieLens dataset described in figure 2 consist of a chronologically ordered set of user-item pairs in the form (u, i). Since we intend to retain only positive feedback on some models, movie ratings below the maximum rating 5 are excluded. To be fair with comparing results, we test all the models in the same setting for the dataset.

We split the dataset using a time-based criterion, with 20% of the data used for training to avoid the cold start problem and the remaining 80% for testing our results. In the case of grid search (grid search is the process of performing hyper parameter tuning in order to determine the optimal values for a given model), the percentage of the training dataset is the same as above, but for the validate dataset, the percentage is set to 30%. In both cases, the user has the ability to change these percentages in the input file as explained in the README file attached with the code.

5 Results

After trying different parameters for each model, we have reached the parameters that lead to the best results. We measured their accuracy using recall@N and DCG@N at cut-offs $N \in \{1, 5, 10, 50, 100\}$ and MRR.

For both NN models (user based and item based), the results have not changed at all with different values of the neighborhood parameter due to the sparsity of the frequency and similarity matrices, but this calls for further investigation that will be pursued later on.

Tables 3 and 4 show the results for NN user-based model and NN item-based model respectively.

Focusing on NN approaches, the item-based one has shown to outperform user-based one in the recall and MRR metrics. Besides their very good results, item-based approaches have other advantages. It is able to produce relevant predictions as soon as a user has rated one item. In addition to that, such models are also appropriate for recommending items in catalogues even when no information about the current user is available, since it can present to a user the nearest neighbours of any item the user is currently interested in. We expected the NN item-based approach to take less time than the user based one because the number of users is greater than the number of items, but the result is the opposite. The reason behind such

	$K=5$	$K=10$	$K=20$	$K=50$	$K=100$
MRR	0.016	0.016	0.016	0.016	0.016
Recall@1	0.006	0.006	0.006	0.006	0.006
Recall@5	0.0145	0.014	0.014	0.014	0.014
Recall@10	0.0247	0.0247	0.0247	0.0247	0.0247
Recall@50	0.12	0.12	0.12	0.12	0.12
Recall@100	0.22	0.22	0.22	0.22	0.22

Table 4: Evaluation results for nearest neighbor, item based model

result is the sparsity of the matrices and the similarity measure (cosine one) we are considering. For item based, it's more likely to find common users that used the same item, while for user based, it's less likely to find common items for two users due to the sparsity of the matrices. So, even if there are more users, the computations are less because there are more zeros in the similarity measure.

Tables 5, 6, and 7 list the overall results for the recall, DCG and MRR evaluation metrics respectively for different algorithms. For the NN algorithms, we take the average across different values of $K = [5, 10, 20, 50, 100]$

Focusing on the MF models, the MF with negative feedback model gives better performance than vanilla MF (as expected). The main reason behind this is the fact that MF with negative feedback considers l negative items for every positive observation. The one with BPR approach is still under testing and analysing that's why its results are not included.

According to the results in tables 5, 6, and 7, it's clear that the Matrix Factorization with negative feedback model defeat all models due to the fact that negative feedback is being considered. On the contrary to that model, Random model gives the worst results because it considers random recommendations for users. In addition, the results show that the most popular model is performing better than all other approaches except MF with negative feedback in the MRR and recall evaluation metrics. The main reason behind this is that the most popular model recommends the most popular movies to the user (and the user usually watches the popular movies).

In summary, according to tables 5 and 7 the evaluation results from best to worst are as follows: MF with negative feedback, most popular, MF, NN item-based, NN user-based, and random model for the MRR and Recall metrics. While for the DCG metric, the results were totally different; according to figure 6 the results from best to worst are: MF with negative feedback, MF, NN user-based, NN item-based, most popular, and random model. The main reason behind such difference is the fact that DCG measures ranking quality and not accuracy.

	Random	MP	MF	MFNF	KNNU	KNNI
Recall@1	0.00038	0.00875	0.00387	0.00221	0.00623	0.00623
Recall@5	0.00170	0.03586	0.01714	0.05868	0.01407	0.01407
Recall@10	0.00306	0.06737	0.03276	0.10701	0.02519	0.02519
Recall@50	0.01579	0.21800	0.13212	0.28468	0.09258	0.11363
Recall@100	0.031577	0.33207	0.23051	0.37954	0.18685	0.21233

Table 5: Evaluation results for Recall metric

	Random	MP	MF	MFNF	KNNU	KNNI
DCG@1	0.00038	0.00087	0.0043	0.00256	0.00397	0.00024
DCG@5	0.00170	0.00365	0.01951	0.05767	0.01016	0.00187
DCG@10	0.00307	0.00694	0.03689	0.11574	0.01728	0.00347
DCG@50	0.01588	0.02385	0.15113	0.42441	0.06184	0.02261
DCG@100	0.03194	0.03786	0.26694	0.66884	0.06906	0.04359

Table 6: Evaluation results for DCG metric

6 Conclusion and Future Work

6.1 Conclusion

RS are an effectual technology for advising users about what products/items to purchase. These systems help users to find items they like or they want to buy from a business; as well as business by generating more sales. RS are rapidly becoming a crucial tool in E-commerce on the Web as they are being stressed by the huge volume of user data. This report discussed our python framework for RS. The framework is designed specifically to deal with stream-based recommendation scenarios. Our work will attempt be contributing to the community of RS by providing: an implementation to a set of state-of-the-art recommendation algorithms in python, an easy comparison among recommendation algorithms in terms of evaluation metrics, and an easy platform to add RS models.

6.2 Future Work

Many different adaptations, models, and experiments have been left for the future due to lack of time. Future work concerns deeper trying and analyzing different methods, like content-based filtering. Content-based filtering offers solutions to the limits of collaborative filtering, i.e, with the case of a new user or a new item, items can be recommended for a user based on the features of the items and the personal information provided by the user. These complementary approaches motivate the design of hybrid systems. Moreover, web services suffer from the collecting of huge amount of data and from recommending billions of items to users. Applying recommending models on large datasets can limit the performance of the system. For

	Random	MP	MF	MFNF	KNNU	KNNI
MRR	0.00278	0.03206	0.01793	0.03819	0.01596	0.01637

Table 7: Evaluation results for MRR metric

large datasets, we can work on the scalability problem of recommendation systems.

References

[Dia+12a] Ernesto Diaz-Aviles et al. “Real-time top-n recommendation in social streams”. In: *Proceedings of the sixth ACM conference on Recommender systems*. 2012, pp. 59–66.

[Dia+12b] Ernesto Diaz-Aviles et al. “What is happening right now... that interests me? online topic discovery and recommendation in twitter”. In: *Proceedings of the 21st ACM international conference on Information and knowledge management*. 2012, pp. 1592–1596.

[Eks19] Michael D Ekstrand. “The LKPY Package for Recommender Systems Experiments”. In: (2019).

[Fri+17] Erzsébet Frigó et al. “Alpenglow: Open source recommender framework with time-aware learning and evaluation”. In: (2017).

[Guo+15] Guibing Guo et al. “LibRec: A Java Library for Recommender Systems.” In: *UMAP Workshops*. Vol. 4. 2015.

[JJK18] Michael Jugovac, Dietmar Jannach, and Mozhgan Karimi. “Streamingrec: a framework for benchmarking stream-based news recommenders”. In: *Proceedings of the 12th ACM Conference on Recommender Systems*. 2018, pp. 269–273.

[Luo18] Shuyu Luo. “Introduction to Recommender System”. In: 2018. URL: https://towardsdatascienc com/intro-to-recommender-system-collaborative-filtering-64a238194a26.

[MJ09] Catarina Miranda and Alıpio Mário Jorge. “Item-based and user-based incremental collaborative filtering for web recommendations”. In: *Portuguese Conference on Artificial Intelligence*. Springer. 2009, pp. 673–684.

[03] “MovieLens 1M Dataset”. In: 2003. URL: https://grouplens.org/datasets/movielens/1m/.

[Nau18] dewang Nautiyal. “Top 5 best Programming Languages for Artificial Intelligence field”. In: 2018. URL: https://www.geeksforgeeks.org/top-5-best-programming-languages-for-artificial-intelligence-field/.

[Pap+05] Manos Papagelis et al. “Incremental collaborative filtering for highly-scalable recommendation algorithms”. In: *International Symposium on Methodologies for Intelligent Systems*. Springer. 2005, pp. 553–561.

[Ren+12] Steffen Rendle et al. “BPR: Bayesian personalized ranking from implicit feedback”. In: *arXiv preprint arXiv:1205.2618* (2012).

[VJG15] João Vinagre, Alıpio Mário Jorge, and João Gama. “Collaborative filtering with recency-based negative feedback”. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. 2015, pp. 963–965.

[VJG14] João Vinagre, Alıpio Mário Jorge, and João Gama. “Fast incremental matrix factorization for recommendation with positive-only feedback”. In: *International Conference on User Modeling, Adaptation, and Personalization*. Springer. 2014, pp. 459–470.

7 Appendix

Algorithm 1 Most Popular

```
 1: procedure BATCH(training set: D)
 2:     $for < u, i > \in D$ do:
 3:         Calculate the frequency of each item
 4:         Sort items' frequency in a sorted score

 5: procedure INCREMENTAL(testing set: D)
 6:     for each test user $(u, i)$:
 7:         get_recommendations(u)
 8:         update_model(u,i)


 9: procedure GET_RECOMMENDATIONS(u)
10:     return the sorted score

11: procedure UPDATE_MODEL(u,i)
12:     increment the frequency of item $i$
```

Pseudo Code for K Nearest Neighborhood Item Based $(KNNI)$ Approach Algorithm: Let F (n $\times$ n) where n is the number of items in the dataset D be the frequency matrix with the number of users that evaluate each pair of items, i.e F(i,j)= number of users that evaluate both items i and j. This matrix is used as an auxiliary matrix to update S. S (n $\times$ n) is the similarity matrix with cosine similarity measure (introduced above), i,e S(i,j)=cosine similarity(i,j)=$cos(\vec{i}, \vec{j})$.

Algorithm 2 kNNI

```
    Input: K: number of neighbors
 1: procedure BatchTraining(training dataset)
 2:     Compute_frequency_ matrix(F)
 3:     Compute_similarity_matrix(S)

 4: procedure IncrementalTraining(testing dataset)
 5:     for each test user $(u, i)$:
 6:         get_recommendations(u)
 7:         update_frequency_matrix(F(u,i))
 8:         update_similarity_matrix(S(u,i))

 9: procedure Compute_frequency_matrix((F))
10:     for i in range(n):
11:         for j in range(n):
12:             F(i,j)= number of users evaluated items i and j

13: procedure Compute_similairty_matrix((S))
14:     for i in range(n):
15:         for j in range(n):
16:             S(i,j)= cosine_similarity(i,j)

17: procedure get_recommendations(u)
18:     determine the activation weight of each item never seen by u s.t.
            $w(i)= \frac{\sum \text{items in the neighborhood that was evaluated by the active user S[item,i]}}{\sum \text{(K) items in the neighborhood S[item,i]}}$
19:     note that, the activation weight of the items that are seen by the user $u$ will be equal to the lowest
    activation weight of unseen items.

20: procedure update_frequency_matrix(F(u,i))
21:     Calculate the user history(u)
22:     increment F(i,i) by one
23:     for every item in the user history: :
24:         increment F(item,i) by one
25:         increment F(i,item) by one
26: procedure update_similarity_matrix(S(u,i))
27:     Calculate the user history(u)
28:     increment S(i,i) by one
29:     for every item in the user history: :
30:         increment S(item,i) by one
31:         increment S(i,item) by one
```

Algorithm 3 SGD

Input: Feedback matrix **R**, training set **D**, regularization parameters λ_u,λ_i, λ_j
, learning rate η , number of features **feat**, number of iterations **iters**, reservoir length **res**
Output: Optimal user features matrix **P**, optimal item feature matrix **Q**

```
 1: Initialize P and Q e.g., randomly or $\sim N(0, 0.1)$
 2: for k $\leftarrow$ 1 to iters do
 3:     $for(u, i) \in D$ do:
 4:         update reservoir (u,i)
 5:         select a user-item pair (u,i) from R uniformly at random
 6:         sample 59 negative pairs for u from the reservoir
 7:         compute the distances $\delta_{uijb}$ for each pair $p_b = ((u, i), (u, j_b)) \in P, b = 1...59$
 8:         Sample a pair $p = ((u, i), (u, j))$ from the chosen negative pairs with probability proportional
    to its informativeness: $1/\delta_{uijb}$
 9:         $y_{uij} \leftarrow sign(x_{ui} - x_{uj})$
10:         $A_u \leftarrow A_u + \eta y_{uij}(B_i - B_j) - \eta\lambda_w Au$
11:         $B_i \leftarrow B_i + \eta y_{uij} A_u - \eta\lambda_{H^+} Bi$
12:         $B_j \leftarrow B_j + \eta y_{uij}(-A_u) - \eta\lambda_{H^-} Bj$
13:         $\eta = \alpha.\eta$
```

Algorithm 4 MF

Input: Regularization parameters λ; Learning rate η ; Number of features (feat), number of batch iterations (iters)

```
1: procedure INITIALIZATION
2:     for u ∈ Users(D) do:
3:         $A_u \leftarrow Vector(size : feat) A_u \sim N(0, 0.1)$
4:     for i ∈ Items(D) do:
5:         $B_i \leftarrow Vector(size : feat) B_i \sim N(0, 0.1)$

6: procedure BATCH SGD(training set: D)
7:     for count ← 1 to iters do:
8:         $for < u, i > \in D$ do:
9:             update_model(u,i)

10: procedure INCREMENTAL_SGD(testing set: D)
11:     for each test user $(u, i)$:
12:         get_recommendations(u)
13:         update_model(u,i)
14: procedure GET_RECOMMENDATIONS(u)
15:     Calculate the score for each item st score(item)=$A_u \times B_i$

16: procedure UPDATE_MODEL(u,i)
17:     $error = 1 - A_u B_i^\intercal$
18:     $A_u \leftarrow A_u + \eta(err_{ui} Bi\text{-}\lambda Au)$
19:     $B_i \leftarrow B_i + \eta(err_{ui} Au\text{-}\lambda Bi)$
```

Algorithm 5 MFNF

Input: Regularization parameters λ; Learning rate η ; number of batch iterations (iters), number of negative items (l)

```
1:  procedure Initialization
2:      for u ∈ Users(D) do:
3:          $A_u \leftarrow Vector(size : feat) A_u \sim N(0, 0.1)$
4:      for i ∈ Items(D) do:
5:          $B_i \leftarrow Vector(size : feat) B_i \sim N(0, 0.1)$

6:  procedure Batch SGD(training set: D)
7:      for count ← 1 to iters do:
8:          $for < u, i > \in D$ do:
9:          update_user_item_models_with_NF(u,i)

10: procedure Incremental_SGD(testing set: D)
11:     for each test user $(u, i)$:
12:         get_recommendations(u)
13:         update_user_item_models_with_NF(u,i)
14: procedure get_recommendations(u)
15:     Calculate the score for each item st score(item)=$A_u \times B_i$

16: procedure update_user_item_models_with_NF(u,i)
17:     Q=Queue()
18:     for $k \leftarrow 1$ to $min(l, \#Q)$ do:
19:         $item_j \leftarrow$ dequeue(Q)
20:         $error = 0 - A_u B_j^\intercal$ (0 and not 1 because the item is negative)
21:         $A_u \leftarrow A_u + \eta(err_{uj} Bj\text{-}\lambda Au)$
22:         enqueue(Q, j)
23:     update_model(u,i)
24:     if i in Q:
25:         remove(Q, i)
26:     enqueue(Q, i)
27: procedure update_model(u,i)
28:     $error = 1 - A_u B_i^\intercal$
29:     $A_u \leftarrow A_u + \eta(err_{ui} Bi\text{-}\lambda Au)$
30:     $B_i \leftarrow B_i + \eta(err_{ui} Au\text{-}\lambda Bi)$
```

Algorithm 6 MFNFBPR

Input: Regularization parameters for users, positive items and negative items are respectively $\lambda_W, \lambda_{H^+}, and \lambda_{H^-}$; Learning rate η ; Learning rate schedule α; number of incremental iterations(incremental_ites); reservoir_length; paramter to perform model updates (update_model_counter); number of batch iterations (iters)

```
 1: procedure INITIALIZATION
 2:     for u ∈ Users(D) do:
 3:         $A_u \leftarrow Vector(size : feat) A_u \sim N(0, 0.1)$
 4:     for i ∈ Items(D) do:
 5:         $B_i \leftarrow Vector(size : feat) B_i \sim N(0, 0.1)$

 6: procedure BATCH SGD(training set: D)
 7:     for count ← 1 to iters do:
 8:         $for < u, i > \in D$ do:
 9:         update_reservoir(u,i)
10:         update_model(u,i)

11: procedure INCREMENTAL_SGD(testing set: D)
12:     for each test user $(u, i)$:
13:         get_recommendations(u)
14:         update_reservoir(u,i)
15:         counter=counter+1
16:         if counter==update_model_counter:
17:             update_model()
18:             counter=0
19: procedure GET_RECOMMENDATIONS(u)
20:     Calculate the score for each item st score(item)=$A_u \times B_i$

21: procedure UPDATE_RESERVOIR(u,i)
22:     if $len(reservoir)$ >=reservoir_length:
23:         remove a random pair from the reservoir
24:     reservoir.append((u,i))

25: procedure UPDATE_MODEL(u,i)
26:     for count ← 1 to incremental_iters do:
27:         Select a user-item pair $(u, i)$ from $R$ uniformly at random
28:         sample 59 negative pairs for u from the reservoir
29:         compute the distances $\delta_{uijb}$ for each pair $p_b = ((u, i), (u, j_b)) \in P, b = 1...59$
30:         Sample a pair $p = ((u, i), (u, j))$ from the chosen negative pairs with probability proportional
    to its informativeness: $1/\delta_{uijb}$
31:         $y_{uij} \leftarrow sign(x_{ui} - x_{uj})$
32:         $A_u \leftarrow A_u + \eta y_{uij}(B_i - B_j) - \eta\lambda_w Au$
33:         $B_i \leftarrow B_i + \eta y_{uij} A_u - \eta\lambda_{H^+} Bi$
34:         $B_j \leftarrow B_j + \eta y_{uij}(-A_u) - \eta\lambda_{H^-} Bj$
35:         $\eta = \alpha.\eta$
```