information

Article

Sequeval: An Offline Evaluation Framework for Sequence-Based Recommender Systems

Diego Monti [1,*], Enrico Palumbo [1,2,3], Giuseppe Rizzo [4] and Maurizio Morisio [1]

1 Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Turin, Italy; enrico.palumbo@ismb.it (E.P.); maurizio.morisio@polito.it (M.M.)
2 Istituto Superiore Mario Boella, Via Pier Carlo Boggio 61, 10138 Turin, Italy
3 EURECOM, Campus SophiaTech, 450 Route des Chappes, 06410 Biot, France
4 LINKS Foundation, Via Pier Carlo Boggio 61, 10138 Turin, Italy; giuseppe.rizzo@linksfoundation.com
* Correspondence: diego.monti@polito.it; Tel.: +39-011-0907087

Received: 15 April 2019; Accepted: 7 May 2019; Published: 10 May 2019

Abstract: Recommender systems have gained a lot of popularity due to their large adoption in various industries such as entertainment and tourism. Numerous research efforts have focused on formulating and advancing state-of-the-art of systems that recommend the right set of items to the right person. However, these recommender systems are hard to compare since the published evaluation results are computed on diverse datasets and obtained using different methodologies. In this paper, we researched and prototyped an offline evaluation framework called Sequeval that is designed to evaluate recommender systems capable of suggesting sequences of items. We provide a mathematical definition of such sequence-based recommenders, a methodology for performing their evaluation, and the implementation details of eight metrics. We report the lessons learned using this framework for assessing the performance of four baselines and two recommender systems based on Conditional Random Fields (CRF) and Recurrent Neural Networks (RNN), considering two different datasets. Sequeval is publicly available and it aims to become a focal point for researchers and practitioners when experimenting with sequence-based recommender systems, providing comparable and objective evaluation results.

Keywords: evaluation framework; offline evaluation; sequence; sequence-based recommender systems; recommender systems; metrics

1. Introduction

Recommender Systems (RSs) are software tools capable of suggesting items to users according to their preferences [1]. A popular recommendation technique, called Collaborative Filtering (CF), consists of learning users' preferences by only relying on their interactions with the items available in a catalog. For example, using a nearest-neighbor search (memory-based CF) or a machine learning model (model-based CF), it is possible to select the most relevant items for each user who has interacted enough with the system [2]. An alternative approach to this problem is represented by content-based RSs, which can generate suggestions by matching users' profiles with the features of the items [3,4]. Another family of recommendation methods proposed in the literature is represented by hybrid algorithms, capable of combining both collaborative and content-based filtering for mitigating the individual weaknesses of the previous techniques [5].

While these recommendation approaches usually guarantee interesting results in traditional domains, such as movie recommendation, they are not capable of capturing the temporal evolution of users' preferences [6]. For example, different authors [7–9] argue that movies watched recently provide more useful information about a certain user than those she consumed in a distant past. It is, in fact, reasonable to assume that a recent item may have a high influence on the choice of the next one.

Therefore, a RS that exploits sequential data for predicting the sole next item that will be consumed by a user can be defined as *sequential recommender* [10]. Several works related to sequential recommenders are available in the literature. For example, Zhou et al. [11] exploited a sequential pattern mining algorithm for recommending which page to visit next in a website, while Rendle et al. [8] relied on Markov chains for suggesting products considering previous purchases. Recently, He et al. [9] designed a recommender system capable of modeling how users' interests evolve over time. While all these methods usually consider user preferences observed during the training phase as sequences, no temporal ordering is available at recommendation time, as only one item, or a list of items ranked by relevance, is suggested to users. Because of the popularity of CF techniques, most of sequential recommenders are based on such approaches [12], but in principle it is also possible to design systems capable of analyzing sequences according to content-based methods.

In general, even if the problem of creating a sequence of words starting from an initial one is a well-known task inside the natural language processing community [13], the idea of creating personalized sequences of items is less widespread in the context of RSs [14]. For this reason, it would be interesting to be able to exploit the temporal ordering not only during the training phase but also for generating sequences of recommended items, such as in the task of language modeling. Some solutions to this problem have already been proposed in industry, and also few researchers have discussed how to automatically construct music playlists [15] or suggest sequences of points-of-interest to tourists [16] starting from seed items.

However, early studies conducted in this field lack of a common definition of the problem that they are trying to address. For example, *session-based* RSs only consider the last session of the current user [17], while *sequence-aware* recommenders also exploit the history of past sessions [12] and they can be considered equivalent to sequential recommenders. Furthermore, it is not clear if item repetitions are allowed in the suggestions or not.

In this work, we argue that it is possible to consider RSs capable of creating personalized sequences of an arbitrary length as a generalization of a sequential recommender because the latter is only able of creating sequences of length *one*. In contrast to traditional RSs that usually create lists of items ranked by relevance, in the following we will define a recommender that exploits a temporal dimension both in the training and in the generation phase as a *sequence-based* recommender, as it observes and suggests sequences of items meant be consumed in a particular order.

On the other side, several evaluation protocols and metrics for analyzing novel RSs via offline experiments are available, to capture the different aspects of the recommendation algorithm [18]. However, the lack of a standardized way of performing such *in vitro* experiments leads to results that are often incomparable [19]. To the best of our knowledge, no evaluation framework for sequence-based RSs has already been proposed. The motivating hypothesis of this study is that in the context of sequence-based RSs, traditional evaluation metrics need to be computed at the level of sequences instead of the level of users.

In our view, an evaluation framework consists of a methodology for performing an experimental comparison, a set of metrics, and a software tool that implements them. Therefore, the main aim of this work is to address the following research questions and to introduce an offline evaluation framework for sequence-based RSs that we called Sequeval.

RQ1 What is the formal definition of a sequence-based recommender system?
RQ2 How already established metrics can be extended and adapted for evaluating a sequence-based recommender system?
RQ3 Against which baselines a sequence-based recommender system can be compared?

Because our evaluation approach is agnostic with respect to the implementation details of the algorithms under analysis, it can be successfully exploited to also assess the performance of systems based on alternative recommendation methods [20] or dealing with unconventional categories of items, for example 3D movies [21] or cultural digital contents [22], especially if they are supposed to be consumed by users in a sequential order.

Besides, by openly releasing a Python implementation of Sequeval, we aim to encourage the use of the proposed framework as an attempt to standardize the evaluation of sequence-based RSs, mitigating the comparability problem in RSs research.

The remainder of this paper is organized as follows: in Section 2 we review related works; in Section 3 we present the mathematical definition of a sequence-based recommender system; in Section 4 we introduce Sequeval by describing its evaluation protocol, metrics, and implementation details; in Section 5 we perform an empirical analysis of the framework with two different datasets. Finally, in Section 6, we formulate our conclusions and we outline future works.

2. Related Work

In this section, we distinguish among works related to RSs capable of exploiting sequences of items (Section 2.1), evaluation protocols and metrics for performing an offline comparison (Section 2.2), and their experimental reproducibility (Section 2.3).

2.1. Sequential Recommenders

Several examples of sequential recommenders are available in the literature [12]. Zhou et al. [11] proposed a web recommender system based on a sequential pattern mining algorithm. The recommender is trained with the access logs of a website and its goal is to predict the pages that are likely to be visited by a certain user, given her previously visited pages. The authors proposed to store the model in a tree-like structure, relying on a technique originally designed for matching substrings over a finite alphabet of characters. A recommendation is then created by matching the sequence of pages already visited by the target user with the sequences previously analyzed by the algorithm.

In the context of market basket analysis, it is also possible to exploit the sequence of previous transactions to predict what a customer is going to buy next [23]. Rendle et al. [8] proposed a method based on personalized transition graphs over Markov chains, while Wang et al. [10] designed a recommender capable of modeling both the sequential information from previous purchases and the overall preferences by a hybrid representation.

Bellogín and Sánchez [24] proposed a similarity metric designed to compare users in the context of CF RSs. This metric takes into account the temporal sequence of users' ratings to identify common behaviors. The authors argue that it is possible to consider a sequence of items as a string, where each character represents an item, and compare them using the longest common subsequence algorithm [25].

Recently, He et al. [9] introduced the concept of *translation-based* recommendation. While a traditional RS only considers the pairwise interactions between items and users, their idea is to model a third-order relationship among a user, the items she interacted with in the past, and the item she is going to visit next. Each user can be represented as a vector in a transition space: given the current item, it is possible to compute where the next one will be located. At recommendation time, it is possible to generate a list of suggested items by relying on a nearest-neighbor search.

On the other hand, the task of generating recommendations of sequences was already discussed and presented in a seminal work by Herlocker et al. [14]. The authors suggested that it would be intriguing to be able to suggest, in the music domain, not only the songs that will be probably liked by a certain user, but also a playlist of songs that is globally pleasing. Moreover, they also proposed to apply this recommendation methodology in the context of scientific literature, where it is necessary to read a sequence of articles to become familiar with a certain topic of interest.

The former problem was later addressed by Chen et al. [15], who designed and implemented a recommender system capable of generating personalized playlists by modeling them as Markov chains. Their algorithm is capable of learning, from a set of training playlists, how to represent each song as a point in a latent space. Then, starting from a seed song, it is possible to create a playlist of an arbitrary length by repeatedly sampling the transition probabilities between adjacent songs. The resulting playlist is personalized because of the chosen seed. Furthermore, each user can influence the generation process by specifying some parameters: for example, a user might be more interested in popular songs, while another one in songs that are strictly related to the given seed.

Another typical application for a sequence-based recommender is the *next* point-of-interest (POI) prediction problem [12]. Given some training sequences of previously liked geographical locations, this task consists of predicting a sequence of venues that is pleasing for a given user. Feng et al. [16] proposed an algorithm capable of creating sequences of POIs that have not been already visited. The authors developed a Metric Embedding algorithm that captures both the sequential information and individual preference. Such metric is then exploited to create a Markov chain model capable of representing the transition probabilities between a given POI and the next one. The key features that are implicitly considered in the embedding creation phase are the conceptual similarity and the geographical distance of the analyzed venues.

A different line of research is represented by recommenders capable of analyzing sequences of multimedia objects. For example, Albanese et al. [26] proposed a hybrid recommender system for retrieving multimedia content based on the theory of social choice and capable of exploiting, among other signals, the implicit browsing preferences of its users. Later, the authors of [27] introduced a multimedia recommendation algorithm capable of combining semantic descriptors and usage patterns. The proposed approach can manage different media types and it enables users to explore several multimedia channels at the same time. Possible applications of such technologies are represented by browsing tools for virtual museums [28] and recommenders of cultural heritage sites [29].

Differently from the previously reported works, our main aim is not to propose a novel recommendation algorithm, but a general method to standardize how the sequences of recommended items are generated and evaluated in an offline setting.

2.2. *Offline Evaluation*

To the best of our knowledge, the first survey that deals with the problem of evaluating a recommender system was conducted by Herlocker et al. [14]. In their work, the authors discuss when it is appropriate to perform an offline evaluation and when it is necessary to carry on an online, or in vivo, experiment. The former is particularly useful to select a small set of potentially good candidates that will be further compared in a real scenario. However, to be complete and trustworthy, such a evaluation needs to rely on a set of well-defined metrics, that should be able to capture all characteristics of the recommended items.

They review several accuracy metrics usually exploited by different authors and they classify them into three categories: predictive accuracy metrics, classification accuracy metrics, and rank accuracy metrics. These groups are strictly related to the purpose of the recommender system: predicting a rating for each user-item pair, identifying an item as appropriate or not for a user, and creating an ordered list of recommended items for a user. Furthermore, the authors suggest that it is necessary to avoid relying only on accuracy to draw a reliable conclusion: for this reason, they also review and discuss the metrics of coverage, learning rate, novelty, serendipity, and confidence.

Gunawardana and Shani [18] proposed a set of general guidelines for designing experiments with the purpose of evaluating RSs. Such experiments can be classified as offline trails, user studies, or online analysis that involve a live system. Several properties of a recommender system can be evaluated: for example, the most common ones are user preference, prediction accuracy, coverage, and utility. The authors argue that the possibility of measuring these properties is strongly influenced by the kind of study and, in the most extreme scenario, some of them cannot be obtained. For example, it is very difficult to measure users' preference in an offline setting.

For each property, different commonly exploited metrics are presented and discussed. Even if the main metrics proposed for evaluating the most popular properties are widely known and understood, usually there is little agreement about the most appropriate metrics for characterizing the least common properties. For example, several definitions, and several metrics, related to the property of utility are available in the literature. The authors also point out that a key decision of offline experiment design is the splitting protocol because this choice will greatly influence the final outcome of the measures.

2.3. Experimental Reproducibility

Different authors analyzed the experimental reproducibility of offline evaluations. For example, Jannach et al. [19] compared several recommendation algorithms in an offline experiment, analyzing their performance by relying on a comprehensive evaluation framework. The authors considered different splitting protocols and metrics, designed to characterize both the accuracy, in terms of rating and ranking, and the coverage of the suggested items. The results of the experimental trails suggest that some common algorithms, despite their high accuracy, tend to only recommend popular items that are probably not very interesting for the users of a real system. This problem is related to the popularity biases introduced by the offline evaluation protocol: for this reason, it is not advisable to compare different algorithms by relying only on measures related to their accuracy. In addition, different splitting protocols produce significantly different and non-comparable outcomes.

Bellogín et al. [30] proposed an evaluation framework designed following the methodologies of the Information Retrieval (IR) field. They suggest that the evaluation procedures available in IR are widespread: for this reason, they could be successfully exploited by the RS community to create a shared evaluation protocol based on ranking, as this setting is more similar to the one of a live system. Unfortunately, three different decisions need to be taken to achieve this goal.

The items considered for the evaluation could be all items available in the dataset, or only the items available in the test set. The non-relevant items for a certain user could be represented by all items in the test set not rated by that user, or by a subset of it with a fixed size. Finally, the global metric could be computed by averaging its value on all users, or on all ratings available in the test set. Different design choices will result in different evaluation protocols and results. The authors also identify two sources of biases in offline evaluations protocols: the sparsity bias and the popularity bias.

Several software tools are available with the purpose of simplifying the process of comparing the performance of recommendation algorithms. They typically include some evaluation protocols and a reference implementation of well-known techniques. Said and Bellogín [31] compared several of these tools to check if their results are consistent. They discovered that the values obtained with the same dataset and algorithm may vary significantly among different frameworks. For this reason, it is not feasible to directly compare the scores reported by these tools, because they are obtained relying on several protocols. The discrepancies reported by the authors are mainly caused by the data splitting protocol, the strategy used to generate the candidate items, and the implementation choices related to the evaluation metrics.

3. Sequence-Based Recommender Systems

Before analyzing the evaluation framework, we introduce the problem of recommending sequences and we provide an answer to RQ1. In a traditional recommender system, users express positive or negative preferences about a certain item. An item may be, for example, a product, a song, or a place. In contrast, we assume that when a user consumes or interacts with an item, she expresses an implicit rating about it. This assumption in the literature goes under the name of *implicit feedback* [14]. Because we are also considering the temporal dimension to build the sequences, each rating is associated with a timestamp that represents the point in time when it was recorded.

Definition 1. *Given the space of items $\mathcal{I}$, the space of users $\mathcal{U}$, the space of timestamps $\mathcal{T}$, a rating $\mathbf{r} \in \mathcal{R}$ is a tuple $\mathbf{r} = (\iota, \upsilon, \tau)$, where $\iota \in \mathcal{I}$ is the item for which the user $\upsilon \in \mathcal{U}$ expressed a positive preference at the timestamp $\tau \in \mathcal{T}$.*

By relying on the set of ratings $\mathcal{R}$ available in the system, it is possible to construct the sequences that will be used to train and to evaluate the recommender. Each sequence only includes the ratings expressed by a single user. On the other hand, each user may produce several sequences.

The concept of *sequence* is similar to the concept of *session* in a traditional web interaction: if two ratings are distant in time more than an interval $\delta\tau$, then they belong to different sequences. Some ratings may be isolated and, for this reason, not part of any sequence. The most appropriate value for $\delta\tau$ depends on the domain: for example, in the POI recommendation scenario, it could be considered of a few hours as reported in [16].

Definition 2. *A sequence* $\mathbf{s} \in \mathcal{S}$ *is a temporally ordered list of ratings* $\langle \mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_n \rangle$ *created by a particular user* $v \in \mathcal{U}$*, i.e., for each* i, $\mathbf{r}_i = (\iota_i, v, \tau_i)$ *and* $\tau_i < \tau_{i+1}$.

In Algorithm 1, we list the procedure for creating the set $\mathcal{S}$, given the set of users $\mathcal{U}$, the set of ratings $\mathcal{R}$, and a time interval $\delta\tau$. Please note that we do not allow the creation of sequences of length *one* because they do not encode a meaningful temporal order.

Algorithm 1 Generation of the set $\mathcal{S}$, given $\mathcal{U}$, $\mathcal{R}$, and $\delta\tau$.

```
Require: $\mathcal{U} \neq \{\varnothing\} \wedge \mathcal{R} \neq \{\varnothing\} \wedge \delta\tau \doteq \tau_i - \tau_j$
 1: $\mathcal{S} \leftarrow \{\varnothing\}$
 2: for all $v \in \mathcal{U}$ do
 3:   $\mathbf{s} \leftarrow \varnothing$
 4:   for all $\mathbf{r}_i \in \mathcal{R}_v : \tau_{i-1} < \tau_i \wedge i > 1$ do
 5:     if $\tau_i < \tau_{i-1} + \delta\tau$ then
 6:       if $\mathbf{s}$ is $\varnothing$ then
 7:         $\mathbf{s} \leftarrow \langle \mathbf{r}_{i-1} \rangle$
 8:       end if
 9:       $\mathbf{s} \leftarrow \mathbf{s} + \langle \mathbf{r}_i \rangle$
10:     else
11:       if $|\mathcal{R}_\mathbf{s}| > 1$ then
12:         $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathbf{s}\}$
13:         $\mathbf{s} \leftarrow \varnothing$
14:       end if
15:     end if
16:   end for
17: end for
18: return $\mathcal{S}$
```

A sequence-based recommender is an RS capable of suggesting a personalized sequence that is built starting from a seed rating $\mathbf{r}_0$, considering the example sequences already available in the system and the specific behavior of a certain user. The seed rating is characterized by a seed item ι_0, a target user v, and an initial timestamp τ_0. The seed item can be represented by any item that belongs to the catalog, but, more in general, it is a point in the space of items $\mathcal{I}$. For example, in the music domain, it could identify not only a particular song, but also an artist, a genre, or a mood. The target user is the user to whom the sequence is recommended, while the initial timestamp represents the point in time in which the recommendation is created. The generated sequence is of a fixed length and it contains exactly k ratings. Please note that if $k = 1$, we are dealing with a sequential RS as defined in [10].

Definition 3. *Given a seed rating* $\mathbf{r}_0 \in \mathcal{R} : \mathbf{r}_0 = (\iota_0, v, \tau_0)$*, and a length* $k \in \mathbb{N}$*, a sequence-based recommender is the function* $\mathrm{sequence} : \mathcal{R} \times \mathbb{N} \to \mathcal{S}$*, i.e.,* $\mathrm{sequence}(\mathbf{r}_0, k) = \langle \mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_k \rangle : \mathbf{r}_i = (\iota_i, v, \tau_i)$.

Most sequence-based recommenders are based on probability models, and therefore they can be interpreted as a sampling function σ applied to the conditional probability $P(\langle \mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_k \rangle | \mathbf{r}_0)$:

$$\text{sequence}(\mathbf{r}_0, k) = \sigma(P(\langle \mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_k \rangle | \mathbf{r}_0)) \tag{1}$$

Using the chain rule, the sequence probability $P(\langle \mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_k \rangle | \mathbf{r}_0)$ can be written as:

$$\begin{aligned} P(\langle \mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_k \rangle | \mathbf{r}_0) =& P(\mathbf{r}_k | \langle \mathbf{r}_0, \ldots, \mathbf{r}_{k-1} \rangle) \cdot \\ & P(\mathbf{r}_{k-1} | \langle \mathbf{r}_0, \ldots, \mathbf{r}_{k-2} \rangle) \cdots P(\mathbf{r}_1 | \langle \mathbf{r}_0 \rangle) \end{aligned} \tag{2}$$

For example, in the case of a Markov chain, each rating depends on the previous one, i.e., $P(\mathbf{r}_k | \langle \mathbf{r}_0, \ldots, \mathbf{r}_{k-1} \rangle) = P(\mathbf{r}_k | \mathbf{r}_{k-1})$:

$$P(\langle \mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_k \rangle | \mathbf{r}_0) = P(\mathbf{r}_k | \mathbf{r}_{k-1}) P(\mathbf{r}_{k-1} | \mathbf{r}_{k-2}) \cdots P(\mathbf{r}_1 | \mathbf{r}_0) \tag{3}$$

Thus, a sequence-based recommender system typically works by learning from a set of sequences $\mathcal{S}_{training}$ the conditional probability of the next rating $\mathbf{r}_k$ to the sequence of previous ones $\langle \mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{k-1} \rangle$, i.e., the factors of the right-hand side of Equation (2). Sampling sequences directly from Equation (2) would require computing the probabilities of all the $|I|^k$ possible sequences, where $|I|$ is the size of the vocabulary of items and k is the length of the sequences. Since this becomes easily computationally unfeasible, we opt for a greedy approach, in which at each step we sample the next most likely item. A sampling function ρ is defined to select a particular next rating from the previous ones at each step:

$$\hat{\mathbf{r}}_k = \rho(P(\mathbf{r}_k | \langle \mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{k-1} \rangle)) \tag{4}$$

A trivial example of ρ is the *argmax* function, which simply selects the most probable next rating. In the following, we will assume that ρ is implemented by a weighted random sampling function.

Algorithm 2 formalizes the procedure for generating a personalized sequence, given a seed rating $\mathbf{r}_0$, and a length k, i.e., it describes the sampling function σ. For k times, the next rating of the recommended sequence is generated using the function *predict*. The function $predict : \mathcal{S} \rightarrow \mathcal{R}$ implements the sampling function ρ and it returns the most probable next rating for the current input sequence. In practice, the sequence-based recommender system can estimate the probability that the next rating of the current sequence will include a particular item at a certain timestamp.

Please note that the greedy procedure described in Algorithm 2 is not the only way to create samples of sequences, and thus, the user of the evaluation framework or the designer of the recommendation method are free to define other ways and strategies for that end.

Algorithm 2 Recommendation of a sequence of length k.

```
Require: $\mathbf{r}_0 \in \mathcal{R} \wedge k > 0$
1: $\mathbf{s} \leftarrow \langle \mathbf{r}_0 \rangle$
2: for $i = 1$ to $k$ do
3:     $\mathbf{r}_i \leftarrow \text{predict}(\mathbf{s})$
4:     $\mathbf{s} \leftarrow \mathbf{s} + \langle \mathbf{r}_i \rangle$
5: end for
6: return $\mathbf{s} - \langle \mathbf{r}_0 \rangle$
```

To compute some metrics that are part of the evaluation framework, it is necessary to know the number of items that are associated with a certain sequence. For this reason, we define the set $\mathcal{I}_\mathbf{s}$ as the set of items that are part of the sequence $\mathbf{s}$, and the set $\mathcal{R}_\mathbf{s}$ as the set of ratings that are part of the sequence $\mathbf{s}$. Therefore, $|\mathcal{I}_\mathbf{s}|$ is the number of distinct items available in $\mathbf{s}$, while $|\mathcal{R}_\mathbf{s}|$ represents the length of $\mathbf{s}$, i.e., the number of ratings available in $\mathbf{s}$.

For instance, we can suppose that the set of ratings $\mathcal{R}$ is equal to $\{(\iota_1, \upsilon_1, \tau_1), (\iota_2, \upsilon_1, \tau_2), (\iota_3, \upsilon_2, \tau_3), (\iota_1, \upsilon_1, \tau_4), (\iota_2, \upsilon_2, \tau_5), (\iota_3, \upsilon_1, \tau_6)\}$. Then, if we assume that the only pair of timestamps that violates the $\delta\tau$ constraint is (τ_4, τ_6), we can create two sequences: $\mathbf{s}_1 = \langle(\iota_1, \upsilon_1, \tau_1), (\iota_2, \upsilon_1, \tau_2), (\iota_1, \upsilon_1, \tau_4)\rangle$, and $\mathbf{s}_2 = \langle(\iota_3, \upsilon_2, \tau_3), (\iota_2, \upsilon_2, \tau_5)\rangle$. The rating $(\iota_3, \upsilon_1, \tau_6)$ is not part of any sequence because it was created at some point in time later than $\tau_4 + \delta\tau$ and we do not have any subsequent rating expressed by υ_1. We also observe that $|\mathcal{I}_{\mathbf{s}_1}| = 2$ and $|\mathcal{R}_{\mathbf{s}_1}| = 3$. We would like to recommend a sequence $\mathbf{s}$ of length *two* to user υ_1 starting from item ι_3 at timestamp $\tau_{\mathbf{s},0}$. In fact, it is not required that the item ι_3 already appeared in the sequences related to user υ_1. A possible solution to this problem is to define $\mathbf{r}_{\mathbf{s},0} = (\iota_3, \upsilon_1, \tau_{\mathbf{s},0})$ and then to recommend $\mathbf{s} = \text{sequence}(\mathbf{r}_{\mathbf{s},0}, 2) = \langle(\iota_2, \upsilon_1, \tau_{\mathbf{s},1}), (\iota_1, \upsilon_1, \tau_{\mathbf{s},2})\rangle$, where $\tau_{\mathbf{s},1}$ and $\tau_{\mathbf{s},2}$ may be used to suggest when consuming the items.

4. Sequeval

Comparing the performance of several recommenders with an experiment that involves a live system is not always feasible or appropriate. For this reason, it is necessary to first perform a preliminary evaluation in an offline scenario [18]. In such a setting, we can assess the performance of the algorithms by comparing with baselines, understanding the strengths and weaknesses of those, and thus avoiding affecting real users negatively in their experience with the system. The results of an offline experiment will be less trustworthy than the ones obtained from an online study because there was no real interaction with the system [32], but they are usually exploited as the first stage in the preparation of the in vivo experimentation.

Even if an offline study is only the first step in the process of evaluating a recommender, it is necessary to rely on a solid evaluation protocol and a set of well-defined metrics that are able to capture all the characteristics of the analyzed algorithm [33]. In the following, we will introduce an offline evaluation framework for sequence-based RSs that we called Sequeval.

Our framework is based on the concept of sequence as formalized in Section 3. First, an initial dataset is transformed into a set of sequences following Algorithm 1. Then, the sequences are split between training and test sets. At this point, one or more external recommenders are plugged into the framework: they are exposed to the training sequences and they are asked to create suggested sequences starting from the same seeds of the test ones. A possible strategy for suggesting additional sequences is described in Algorithm 2, but the user can define other approaches. Finally, considering the recommendations available, the framework can compute different evaluation metrics.

In details, Sequeval is made of an evaluation protocol, presented in Section 4.1, a set of evaluation metrics, described in Section 4.2, and a software implementation that is introduced in Section 4.3.

4.1. Evaluation Protocol

One of the first problems that an evaluation framework should consider is how to split the dataset between the training set and the test set. This task is not trivial, as it will deeply influence the outcome of the experimentation [31]. Since we are dealing with sequences, we need to split the set of sequences $\mathcal{S}$ in a training and test set such that $\mathcal{S} = \mathcal{S}_{training} \cup \mathcal{S}_{test}$.

Several solutions to this problem are possible: a simple but effective one is to perform the splitting by randomly assigning sequences to these sets according to a certain ratio, typically the 80% for training and the 20% for testing. If the number of sequences available is limited, it is necessary to perform a cross-validation. Another possibility is to identify an appropriate point in time and to consider all the sequences created before it as part of the training set, and after it as part of the test set. This protocol simulates the behavior of a recommender introduced at that point in time and it avoids too optimistic results caused by the knowledge of future events [34]. The latter solution can be considered the most reliable one, but if we do not have any temporal information because the sequences have already been created, it is necessary to adopt a random protocol.

More in general, it is impossible to identify a splitting method that is appropriate for every experiment, as it depends on the domain and on the dataset available. For this reason, Sequeval does not impose the adoption of a particular splitting protocol, but the experimenter can choose the most appropriate one.

To compute the metrics that are part of the evaluation framework, the sequence-based recommender is trained with all the sequences $\mathbf{s} \in \mathcal{S}_{training}$. Then, for each test sequence $\mathbf{s} \in \mathcal{S}_{test}$: $\mathbf{s} = \langle \mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_n \rangle$, we predict a recommended sequence $\bar{\mathbf{s}}$ of length k using $\mathbf{r}_1 = (\iota_1, \upsilon, \tau_1)$ as seed rating, i.e., $\bar{\mathbf{s}} = \text{sequence}(\mathbf{r}_1, k)$. Therefore, $\bar{\mathbf{s}}$ is a sequence suggested by the recommender, for the same user and starting from the same item of $\mathbf{s}$. We also define $\mathbf{s}'$ as the reference sequence, i.e., $\mathbf{s}' = \langle \mathbf{r}_2, \mathbf{r}_3, \ldots, \mathbf{r}_n \rangle$ or $\mathbf{s}' = \mathbf{s} - \langle \mathbf{r}_1 \rangle$. The reference sequence is equal to the original sequence, but the first rating is omitted, as it was already exploited for creating the recommended sequence. This procedure is graphically illustrated in Figure 1.

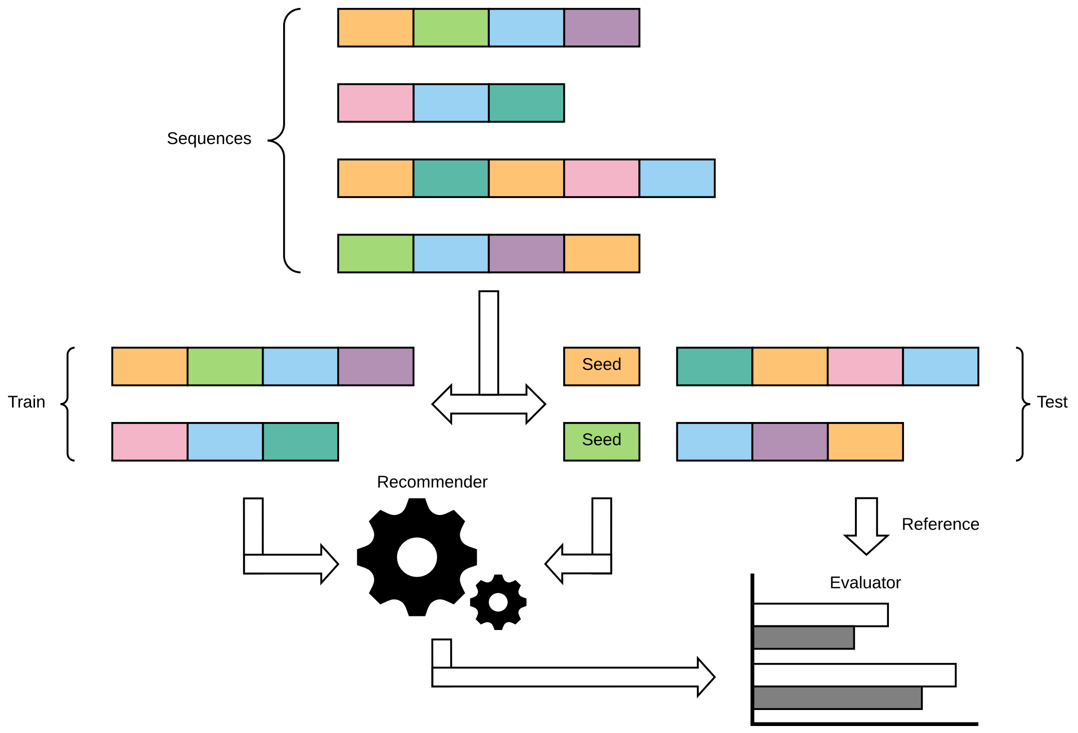

Figure 1. An illustration of the evaluation procedure. First, the set of sequences is split between training and test set. Then, the recommender is trained with the sequences available in the training set. Finally, the recommender is asked to generate a sequence for each seed from the test set; such sequences are compared with the corresponding reference sequences.

4.2. Evaluation Metrics

The second component of Sequeval is a set of eight metrics that we present in the following. In order to address RQ2, we include in such set not only classic metrics such as coverage and precision but also less widespread ones such as novelty, diversity, and serendipity. Furthermore, we introduce the metric of perplexity, as it is explicitly designed for characterizing sequences [35]. In contrast, we decided to avoid measuring recall because it is clear that the number of recommended items is often likely to be much lower than the total number of relevant items.

4.2.1. Coverage

In general, the coverage of a recommender is a measure that captures the number of items in the catalog over which the system can make suggestions [18]. For example, in an online store scenario, it could represent the percentage of products that are recommended to users in a certain period of time. An algorithm with a higher coverage is generally considered more useful because it better helps users to explore the catalog.

We generate a set of recommended sequences considering as seed the first rating of all sequences in the test set $\mathcal{S}_{test}$ for a recommender that suggests sequences of length k. Afterward, we compute the distinct number of items available in the sequences created and we divide the result by the cardinality of the set $\mathcal{I}$.

$$\text{coverage}(k) = \frac{|\bigcup_{\mathbf{s}\in\mathcal{S}_{test}} \mathcal{I}_{\text{sequence}(\mathbf{r}_1,k)}|}{|\mathcal{I}|} \tag{5}$$

This metric expresses the percentage of items that the sequence-based recommender can suggest when generating sequences similar to the ones available in the test set and it is strictly related to its cardinality. This approach is similar to the metric of prediction coverage described by Herlocker et al. [14].

4.2.2. Precision

Precision is a widespread metric in the context of IR evaluation [36] and it represents the fraction of retrieved documents that are relevant. For a traditional recommender system, precision measures the fraction of recommended items that are relevant for a certain user [37]. If we consider a sequence-based recommender, it is necessary to compute this metric for each sequence $\mathbf{s} \in \mathcal{S}_{test}$, instead of each user.

$$\text{precision}(k) = \frac{1}{|\mathcal{S}_{test}|} \cdot \sum_{\mathbf{s}\in\mathcal{S}_{test}} \frac{\text{hit}(\mathbf{s}', \bar{\mathbf{s}})}{\min(|\mathcal{R}_{\mathbf{s}'}|, k)} \tag{6}$$

The function $hit : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{N}$ returns the number of items in $\bar{\mathbf{s}}$ that are also available in $\mathbf{s}'$. If the same item is present in $\bar{\mathbf{s}}$ multiple times, it is considered a hit only if it is repeated also in $\mathbf{s}'$. This is an extension to the traditional definition of precision that also considers the fact that an item may appear multiple times inside a sequence.

The number of relevant items is divided by the minimum number between the length of the reference sequence $|\mathcal{R}_{\mathbf{s}'}|$ and the length of the recommended sequence k. We decided to adopt this solution to avoid penalizing an algorithm that is evaluated considering reference sequences shorter than the recommended sequences.

4.2.3. nDPM

The Normalized Distance-based Performance Metric (nDPM) was originally proposed by Yao in the context of information retrieval [38]. The intuition of the author is that in order to compare a system ranking with a reference user ranking, it is necessary to consider all the possible pairs of items available in the system ranking: they can be agreeing, contradictory, or compatible with respect to the user ranking. We decided to adopt such a metric instead of the Normalized Discounted Cumulative Gain (nDCG) [39] because, in a sequence of recommendations, it is not necessarily true that the first items are more important than the last ones.

$$\text{nDPM}(k) = \frac{1}{|\mathcal{S}_{test}|} \cdot \sum_{\mathbf{s}\in\mathcal{S}_{test}} \frac{2\,\text{pairs}^{-}(\mathbf{s}', \bar{\mathbf{s}}) + \text{pairs}^{u}(\mathbf{s}', \bar{\mathbf{s}})}{2\,\text{pairs}(\bar{\mathbf{s}})} \tag{7}$$

The function $pairs^{-} : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{N}$ returns the number of pairs in the sequence $\bar{\mathbf{s}}$ that are in the opposite order with respect to the reference sequence $\mathbf{s}'$. The function $pairs^{u} : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{N}$ returns the number of pairs in the sequence $\bar{\mathbf{s}}$ for which the ordering is irrelevant, i.e., when at least one of the items is not available in $\mathbf{s}'$ or when at least one of the items is available multiple times in $\mathbf{s}'$. Finally, the function $pairs : \mathcal{S} \rightarrow \mathbb{N}$ returns the number of all possible pairs available in the recommended sequence $\bar{\mathbf{s}}$. The pairs are created without considering the ordering of the items inside a pair: for example, if we have the sequence $\langle a, b, c \rangle$, the possible pairs are (a, b), (a, c), (b, c).

The value of this metric will result close to 1 when the sequences generated by the recommender are contradictory, to 0 when they have the same ranking, and to 0.5 when the ordering is irrelevant because they contain different items. A low precision will imply a nDPM very close to 0.5.

4.2.4. Diversity

The metric of sequence diversity included in this framework is inspired by the metric of Intra-List Similarity proposed by Ziegler et al. [40]. The recommended sequences are considered to be lists of items and the obtained value is not related to their internal ordering. The purpose of this metric is understanding if the sequences contain items that are sufficiently diverse. A higher diversity may be beneficial for the users, as they are encouraged to better explore the catalog [41].

$$\text{diversity}(k) = \frac{1}{|\mathcal{S}_{test}|} \cdot \sum_{\mathbf{s}\in\mathcal{S}_{test}} \frac{\sum_{\forall i,\forall j:0<i<j}^{k} 1 - \text{sim}(\bar{\imath}_i, \bar{\imath}_j)}{k \times (k-1)} \tag{8}$$

The function $sim : \mathcal{I} \times \mathcal{I} \to [-1, 1]$ is a generic similarity measure between two items. This measure may be taxonomy-driven or content-based: for example, a possible content-based similarity measure is the cosine similarity. The resulting value is a number in the interval $[0, 2]$: higher values represent a higher diversity.

4.2.5. Novelty

Vargas et al. [42] suggested that it would be useful to be able to characterize the novelty of the recommendations. They proposed a metric that rewards algorithms capable of identifying items that have a low probability of being already known by a specific user because they belong to the long-tail of the catalog. We have included such metric in our framework to assess whether the items available in the suggested sequences are not too obvious.

$$\text{novelty}(k) = -\frac{1}{|\mathcal{S}_{test}| \times k} \cdot \sum_{\mathbf{s}\in\mathcal{S}_{test}} \sum_{i=1}^{k} \log_2 \text{freq}(\bar{\imath}_i) \tag{9}$$

The function $freq : \mathcal{I} \to [0, 1]$ returns the normalized frequency of a certain item $\imath \in \mathcal{I}$, i.e., the probability of observing that item in a given sequence $\mathbf{s} \in \mathcal{S}_{training}$. We can define the probability of observing the item $\imath$ as the number of ratings related to $\imath$ in the training sequences divided by the total number of ratings available. We also assume that $\log_2(0) \doteq 0$ by definition, to avoid considering as novel items for which we do not have any information, i.e., the items that do not appear in the training sequences.

4.2.6. Serendipity

Serendipity can be defined as the capability of identifying items that are both attractive and unexpected [43]. Ge et al. proposed to measure the serendipity of a recommender by relying on the precision of the generated lists after having discarded the items that are too obvious [33].

To create a list of obvious items, it is possible to exploit a *primitive* recommender that is a recommender only capable of making obvious suggestions. For example, a primitive recommender could be implemented using the Most Popular (MP) baseline, which is defined in Section 4.3. It is reasonable to assume that popular items do not contribute to the serendipity of the recommendations because they are already well known by many users.

By modifying the metric of precision described in Section 4.2.2, it is possible to introduce the concept of serendipity in the evaluation of a sequence-based recommender. In this case, the primitive recommender will always create a sequence of length k that contains the items that are have been observed with the highest frequency in the training set.

$$\text{serendipity}(k) = \frac{1}{|\mathcal{S}_{test}|} \cdot \sum_{\mathbf{s}\in\mathcal{S}_{test}} \frac{\text{hit}(\mathbf{s}', \bar{\mathbf{s}} - \hat{\mathbf{s}})}{\min(|\mathcal{R}_{\mathbf{s}'}|, k)} \tag{10}$$

We define $\hat{\mathbf{s}}$ as the sequence generated by the primitive recommender from the same seed of $\bar{\mathbf{s}}$, i.e., $\hat{\mathbf{s}} = \text{primitive}(\mathbf{r}_1, k)$. Moreover, the sequence $\bar{\mathbf{s}} - \hat{\mathbf{s}}$ contains all the ratings related to the items available in $\bar{\mathbf{s}}$ that are not present in $\hat{\mathbf{s}}$. The resulting value will be a number in the interval $[0, 1]$, lower than or equal to precision. The difference between precision and serendipity represents the percentage of obvious items that are correctly suggested.

4.2.7. Confidence

The metric of confidence reflects how much the system trusts its own suggestions and it is useful for understanding how robust the learned model is [44]. It is usually computed as the average probability that the suggested items are correct. This metric expresses the point of view of the recommender, as the probability is reported by the model. Therefore, the metric is always equal to 1 with the MP recommender, as it is certain of the predictions.

A sequence-based recommender generates the next item of the sequence by considering all the previous items. For this reason, we can interpret the conditional probability of obtaining a certain item, given the sequence of previous ones, as the confidence that the system has in that suggestion.

$$\text{confidence}(k) = \frac{1}{|\mathcal{S}_{test}| \times k} \cdot \sum_{\mathbf{s} \in \mathcal{S}_{test}} \sum_{i=1}^{k} P(\bar{\iota}_i | \bar{\iota}_{i-1}, \bar{\iota}_{i-2}, \dots) \tag{11}$$

We also define $\bar{\iota}_0 \doteq \iota_1$, i.e., the zero-th item of the recommended sequence is its seed item. Therefore, this metric is computed by also considering the probability of obtaining the first item $\bar{\iota}_1$, given the seed item of $\bar{\mathbf{s}}$.

4.2.8. Perplexity

Perplexity is a widespread metric in the context of neural language modeling evaluation [35], typically used to measure the quality of the generated phrases. Because there is a strong similarity between creating a sequence of natural language words and sequence of recommended items given an initial seed, perplexity can be also successfully exploited in this context.

This metric can be defined as the exponential in base 2 of the average negative log-likelihood of the model, i.e., the cross-entropy of the model. For models based on the cross-entropy loss function such as neural networks, the perplexity can also be seen as a measure of convergence of the learning algorithm. Differently from the metric of confidence, the conditional probability $P(\iota_{i+1} | \iota_i, \iota_{i-1}, \dots)$ is computed considering the items of the test sequence $\mathbf{s}$, and not of the recommended sequence $\bar{\mathbf{s}}$. For this reason, it does not express the point of view of the recommender.

$$\text{perplexity} = 2^{-\frac{1}{\sum_{\mathbf{s} \in \mathcal{S}_{test}} |\mathcal{R}_{\mathbf{s}}| - 1} \cdot \sum_{\mathbf{s} \in \mathcal{S}_{test}} \sum_{i=0}^{|\mathcal{R}_{\mathbf{s}}|-1} \log_2 P(\iota_{i+1} | \iota_i, \iota_{i-1}, \dots)} \tag{12}$$

Intuitively, the obtained value represents the number of items from which an equivalent random recommender should choose to obtain a similar sequence. The lower is the perplexity, the better is the system under evaluation. Therefore, the perplexity of a random recommender is equal to $|\mathcal{I}|$. If the performance of the recommender is worse than a random one, the perplexity will be higher than $|\mathcal{I}|$: for example, if only one conditional probability is equal to zero, then perplexity $= +\infty$.

4.3. Implementation

The third component of Sequeval is `sequeval` [45], a Python implementation of the evaluation framework which is publicly available on GitHub (https://github.com/D2KLab/sequeval). This implementation is based on the protocol presented in Section 4.1 and it includes the metrics described in Section 4.2.

In details, `sequeval` is a Python package designed following a modular structure. For each component, we defined an abstract class and then we realized one or more possible implementations to enable software extensibility. For example, we have implemented an input module capable of processing a textual file in a MovieLens-like format (UIRT), but the support to other dataset formats can be easily added. The module representing the recommender system under analysis includes an abstract class that is based on the sequence generation logic formalized in Algorithm 2.

To exploit the proposed framework, it is necessary to realize an implementation of the abstract recommender that must be capable, given the user and the current item of the sequence, of predicting the probabilities for the possible items of being the next one inside the recommended sequence. If the weighted random sampling generation logic is not appropriate, it is possible to override the relative method and to define an alternative recommendation approach.

For obtaining the experimental results, it is necessary to write an evaluation script that relies on this library. We provide a simple evaluation script which can be used to perform different experiments. This script can be easily modified to accommodate novel recommendation techniques.

We also created an extensive test suite achieving the 98% of code coverage for validating the robustness of our implementation and for better supporting future improvements. The total development cost of this Python library could be estimated around 40 h.

For demonstrative purposes, we have implemented four baseline recommenders, which are illustrated in the following and represent our answer to RQ3. These baselines can be interpreted as an adaptation of classic non-personalized recommendation techniques to our sequence-based scenario.

Most Popular The MP recommender analyzes the sequences available in the training set to compute the popularity of each item, i.e., the number of times an item appears in the training sequences. Then, at recommendation time, it ignores the seed rating, and it always creates a sequence that contains the MP item as the first rating, the second MP item as the second rating, and so on. More formally, the probability that the item ι_i will appear in the i-th rating of the sequence is $P(\iota_i) = 1$, where i also represents the position of the item in the ranking of the MP ones.

Random The random recommender simply creates sequences composed of ratings that contain an item randomly sampled from a uniform probability distribution. The seed rating is discarded and the probability of observing the item ι_i is $P(\iota_i) = 1/|\mathcal{I}|$, where $|\mathcal{I}|$ represents the number of items available in the system.

Unigram The unigram recommender can generate sequences that contain ratings with items sampled with a probability proportional to the number of times they were observed in the training sequences. In particular, the probability of observing the item ι_i is equal to the number of ratings containing ι_i divided by the total number of ratings available in the training sequences. As with the previous baselines, the seed rating is ignored during the recommendation phase.

Bigram The bigram recommender estimates the 1-st order transition probabilities among all possible pair of items available in the training sequences. The add-one smoothing technique is exploited to avoid the attribution of a strict zero probability to the pairs that were not observed during the training phase [46]. At recommendation time, the seed rating is exploited for selecting the first item, and then each item will influence the choice of the next one. The probability of sampling item ι_i after item ι_{i-1} is equal to the number of times this transition occurred in the training sequences plus one divided by the total number of transitions available.

5. Experimental Analysis

In this section, we perform an experimental analysis of Sequeval by relying on its implementation described in Section 4.3 for comparing the behavior of the four baselines with a recommender system based on Conditional Random Fields (CRF) [47] and another one that exploits Recurrent Neural Networks (RNN) [48]. The purpose of this comparison is to assess the validity of the framework by conducting an offline evaluation in a realistic scenario. Furthermore, we aim to investigate the efficiency of the proposed approach by analyzing the amount of time required to compute the numerical scores per each recommender system, considering datasets of different sizes.

5.1. Experimental Setup

The main parameters that need to be specified according to our evaluation framework are the $\delta\tau$ value used to generate the sequences, the splitting protocol, and the length of the recommended sequences k. The $\delta\tau$ value and the splitting protocol depend on the dataset and they are reported in Section 5.2. For performing this empirical analysis, we have decided to exploit the 80% of the dataset for training the recommenders and the remaining 20% for testing purposes. The length of the recommended sequences depends on the specifications of the target application: for this evaluation, we have chosen to set $k = 5$.

To compute the metric of diversity, we have selected the cosine similarity among the training sequences as the proximity measure between two items. In fact, we assume that two items are similar if they appear the same number of times inside the same training sequences. Furthermore, we have assumed that if an item is unknown, its similarity with another one is *zero* by definition.

In the following, we provide the rationale for the usage and the implementation details of the two RSs based on CRF and RNN.

CRF We have implemented a CRF-based recommender system using the `CRFsuite` software package (http://www.chokkan.org/software/crfsuite). Since we are interested in predicting an item given the previous one, we have considered to be feature vectors the training sequences without their last rating and as corresponding output vectors the same sequences without their first rating. We have used the gradient descent algorithm with the L-BFGS method [49] as the training technique. We have chosen to generate both the state and the transition features that do not occur in the dataset and we have set the maximum number of iterations allowed for the optimization algorithm to 100.

RNN We have also experimented with a sequence recommender, originally designed for the tourism domain, based on RNN [50] that are specifically meant to deal with sequential data. The hyper-parameters of the network have been optimized through a manual search on the validation set in [50], obtaining: n_layers $=$ 3, dropout $=$ 0.2, learning_rate $=$ 0.0001, n_hidden $=$ 64, and n_epochs $=$ 10. The main difference of RNNs with respect to standard feed-forward neural networks is the presence of a hidden state variable h_t, whose value depends both on the input data presented at time x_t and on the previously hidden state h_{t-1} [48] using loop connections. A typical application of RNNs in neural language modeling is the generation of text by recursively applying a "next word prediction" [51]. In the same spirit, we address the problem of next item prediction. The probability of the next rating given the previous ones $P(\mathbf{r}_k|\langle\mathbf{r}_0, \mathbf{r}_1, \dots, \mathbf{r}_{k-1}\rangle)$ is learned during the training process of the neural network without the need for specifying a particular memory window as in Markov models.

For conducting this experimental campaign, we relied on a machine equipped with two 12-cores Intel Xeon processors (E5-2680 v3 at 2.50 GHz) and 128 GB of RAM. However, note that `sequeval` is a single-threaded application and its memory requirements are actually much lower, around 2.5 GB with the most demanding dataset at our disposal.

5.2. Datasets

We have performed the experimental analysis considering two different datasets, namely Yes.com and Foursquare. The former is related to the musical domain, while the latter deals with check-ins performed at specific POIs.

Because we are interested in modeling sequences, it is important that the temporal information available is actually meaningful. For example, the popular MovieLens datasets [52] cannot be exploited for our purposes because the timestamps are associated with the action of assigning a rating on the platform and not with the action of watching a movie. This hypothesis is supported by the fact that, if we apply Algorithm 1 to the MovieLens 1M dataset with $\delta\tau = 1$ h, we obtain unrealistic sequences with an average length of about 56 movies.

The Yes.com and Foursquare datasets are characterized by a different distribution of their items, i.e., songs and venue categories, as it can be observed from Figure 2. In particular, Foursquare contains few items that are extremely popular, while Yes.com presents a plot that is smoother. This conclusion is numerically supported by the values of entropy [53] obtained for the two distributions, which are 4.95 for Foursquare and 6.75 for Yes.com. Furthermore, the number of sequences available in Foursquare is about 40 times higher with respect to Yes.com. Table 1 summarizes the number of users, items, ratings, and sequences available in these datasets, which are described in detail in the following sections.

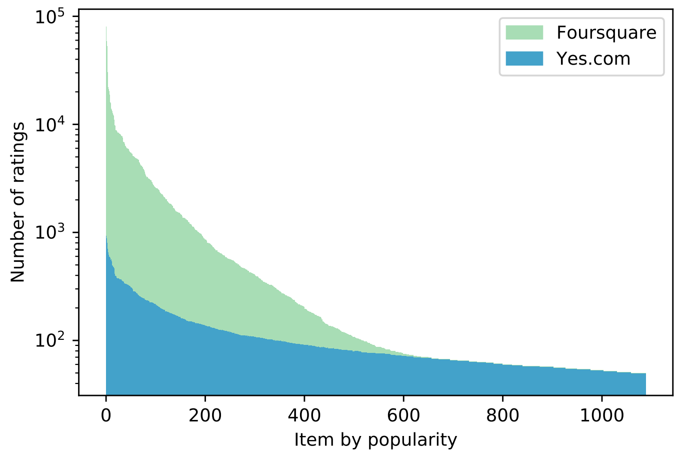

Figure 2. A stacked bar plot with a logarithmic scale representing the number of ratings for each item. Note the different shapes of their long-tail distributions: it is possible to observe that Foursquare has more popular items than Yes.com.

Table 1. The number of users, items, ratings, and sequences after the preprocessing steps.

Dataset	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{S}\rvert$
Yes.com	1	1089	118,022	10,551
Foursquare	44,319	651	1,047,429	400,261

5.2.1. Yes.com

This dataset contains several playlists originally collected by Shuo Chen from Yes.com in the context of his research on Metric Embedding [15]. Such website provided a set of APIs (http://web.archive.org/web/20150316134941/http://api.yes.com) for programmatically retrieving songs aired by different radio stations in the United States. By crawling them in the period from December 2010 to May 2011, he managed to obtain $2,840,553$ transitions. Even if Yes.com is no longer active, the playlist dataset is publicly available (https://www.cs.cornell.edu/~shuochen/lme/data_page.html).

Yes.com does not include the timestamps, but only the playlists. Therefore, we have assumed that each playlist represents a sequence, as defined in our evaluation framework. In this case, it is not necessary to apply Algorithm 1 because the sequences are already available in the dataset in an explicit form. Because a timestamp-based splitting is not feasible, we have selected, for this dataset, a random splitting protocol for dividing the sequences between training and test set.

Since we do not have any information regarding the radio stations, it is necessary to consider the playlists as if they were created by the same user. This approximation is acceptable in the context of sequence recommendation and it is allowed by the evaluation framework. In fact, differently from traditional evaluation approaches, all the metrics that we propose are averaged over the sequences and not over the users.

Because of the computational complexity of the task, we have randomly reduced the complete dataset 10 times its original size and we have pruned the songs that appear less than 50 times.

5.2.2. Foursquare

The second dataset that we have selected for performing the experimental evaluation of the framework is similar to the one described in [50] and it was created following the same protocol.

We collected the check-ins performed by the users of the Foursquare Swarm mobile application (https://www.swarmapp.com) and publicly shared on Twitter from the Twitter API. Then, we retrieved the category of the place associated with the check-in thanks to the Foursquare API. For this reason, the items of the dataset are represented by the venue categories available in the Foursquare taxonomy (https://developer.foursquare.com/docs/resources/categories). The collection phase lasted from October to December 2017.

To avoid exploiting the interactions generated by automated scripts, we have discarded the users that performed multiple check-ins in less than one minute. We have also pruned the check-ins associated with the venue categories that are usually not of interest for a tourist, for example the ones related to workplaces. For generating the sequences more efficiently, we decided to also remove the users that have performed less than 10 check-ins in total.

We have set the $\delta\tau$ parameter of the evaluation framework to 8 h. Regarding the splitting protocol, we have selected the timestamp-based one, considering the timestamp associated with the first rating as the timestamp of the sequence.

5.3. Results

Table 2 summarizes the figures of the evaluation conducted with Yes.com. The MP recommender achieved a fair precision, but at the price of a very low coverage, because its predictions are deterministic. Unsurprisingly, the lowest precision and the highest novelty and diversity are associated with the random recommender. In contrast, the unigram, the bigram, and the CRF recommenders obtained comparable scores of precision, but the bigram is the most appealing of these three techniques, because of its lower perplexity and higher novelty.

Table 2. Overview of the results of the baselines and both CRF and RNN with Yes.com.

Metric	MP	Random	Unigram	Bigram	CRF	RNN
Coverage	0.0046	1.0000	0.9945	1.0000	0.9991	0.9458
Precision	0.0503	0.0090	0.0127	0.0103	0.0190	0.0782
nDPM	0.5007	0.5000	0.5000	0.5000	0.5000	0.4986
Diversity	0.6925	0.9900	0.9815	0.9854	0.9788	0.9052
Novelty	7.2383	10.380	9.7349	10.315	9.8449	9.5762
Serendipity	0.0000	0.0089	0.0107	0.0095	0.0179	0.0706
Confidence	1.0000	0.0009	0.0016	0.0011	0.0020	0.0123
Perplexity	$+\infty$	1089.0	848.96	637.53	747.33	183.49

We can observe that the RNN recommender achieved the highest precision and the lowest perplexity, resulting in be the most promising algorithm for future online experimentations. Its nDPM is slightly lower than 0.5, meaning that the items are usually predicted in the correct order. We can also observe that its serendipity is close to the value of precision: for this reason, it is possible to assume that most of the sequences are not obvious.

Table 3 lists, instead, the results obtained with Foursquare. In this case, the MP recommender system accounted for the highest precision, meaning that the top-5 items are extremely widespread, but, as usual, its coverage is very limited, and it achieved the lowest novelty. On the other hand, the random recommender scored the lowest precision, and the highest coverage and novelty. The differences among the unigram, the bigram, and the CRF recommenders are more striking than in the previous experiment: with this dataset, the unigram accounted for higher precision because of the popularity of some items, while the bigram for the lowest perplexity.

Table 3. Overview of the results of the baselines and both CRF and RNN with Foursquare.

Metric	MP	Random	Unigram	Bigram	CRF	RNN
Coverage	0.0077	1.0000	0.9616	1.0000	0.9677	0.5069
Precision	0.2259	0.0080	0.0774	0.0607	0.0754	0.0962
nDPM	0.4998	0.5000	0.4994	0.4998	0.4993	0.4991
Diversity	0.9194	0.9971	0.9616	0.9777	0.9621	0.9469
Novelty	4.6056	12.300	7.1421	9.0216	7.3710	6.8374
Serendipity	0.0000	0.0060	0.0256	0.0230	0.0252	0.0365
Confidence	1.0000	0.0015	0.0171	0.0140	0.0179	0.0264
Perplexity	$+\infty$	651.00	141.41	122.99	147.49	140.39

The RNN recommender system obtained the second-best precision and perplexity, resulting in a good compromise if we are interested in optimizing both these metrics. Its fair coverage and the low value of serendipity are other hints of the fact that the Foursquare dataset contains few items that are very popular: this characteristic was, in fact, learned and exploited by the recommender.

Finally, we report in Table 4 the amount of time needed for computing the previously described evaluation metrics per recommendation algorithm with the Foursquare and Yes.com datasets. We observe that in the worst case, the evaluation framework was able to conduct an experimental campaign in a few hours. The metric of diversity was the most computationally expensive one because of the time needed to compute the cosine similarity. This fact is particularly evident if we consider the seconds spent to evaluate the random recommender with the Foursquare dataset.

Table 4. The time in seconds required to evaluate different algorithms with our framework. We do not consider the time for training the CRF and RNN models. To improve the efficiency of the framework it is possible to avoid computing the computationally expensive metric of diversity.

Dataset	Diversity	MP	Random	Unigram	Bigram	CRF	RNN
Yes.com	Yes	0.84	4.25	3.97	4.05	963.65	66.04
Yes.com	No	0.78	1.14	0.98	1.05	865.17	60.68
Foursquare	Yes	15.88	1326.31	58.64	101.73	4096.78	1223.26
Foursquare	No	13.75	24.63	13.05	16.78	3989.17	1194.52

As expected, the baseline recommenders are less demanding with respect to the CRF and RNN models. However, this analysis is beyond the scope of this work, as we are only interested in optimizing the evaluation framework. If we do not consider the metric of diversity, we observe that the time required for the evaluation phase is linear with respect to the size of the dataset.

5.4. Discussion

In the following, we will analyze the results of the empirical analysis to justify the answers to the research questions that were provided in Section 3 and in Section 4. In particular, our main aim is to explain why it is necessary to rely on a framework that includes several metrics to evaluate a sequence-based recommender system.

In Section 3 we have introduced the concept of rating and we have defined it considering three different elements: an item, a user, and a timestamp. The idea of associating a user with an item is the basic principle of almost every recommender, while the timestamp is necessary in order to introduce a temporal dimension, and, therefore, the possibility of creating and suggesting sequences, as proposed in RQ1. Nevertheless, we have successfully applied our evaluation framework in an experiment based on the Yes.com dataset, which does not include any user. Even though a more general use case has been considered during its formalization, it is possible to also exploit it in other scenarios, still obtaining an interesting picture of the recommenders.

As described in Section 4.2, our answer to RQ2 is an evaluation framework that includes eight different metrics, capable of capturing the various characteristics of the algorithms available.

For example, even if the precision of the MP recommender system is very high when tested with Foursquare, we can immediately discard it because of its low coverage. In the same way, the interesting values of diversity and novelty achieved by the random recommender are associated with an unacceptable score of perplexity.

In Table 5 we present an interpretation of the metrics available in the framework. These descriptions are meant to offer a human understanding of the results of the offline evaluation. It is worth noticing that these metrics consider only some of the properties of a recommender system [18]. However, it is our opinion that those properties are the most salient ones that can be analyzed in our context, without realizing a live system.

Table 5. A human readable interpretation of the metrics.

Metric	**Interpretation**
Coverage	The percentage of items that are recommended in the evaluation
Precision	The percentage of items that are correctly recommended
nDPM	The correctness of the ordering inside the sequences
Diversity	How diverse are the items inside the sequences
Novelty	How unexpected are the recommended items
Serendipity	The percentage of non-obvious items that are correct
Confidence	The confidence that the recommender has about its predictions
Perplexity	How much the recommender is "surprised" by the test sequences

The different characteristics of the datasets exploited during the empirical analysis are reflected in their respective figures. In particular, while the values of precision obtained by the random recommender in the two experiments are comparable, the figures associated with both MP and unigram methods are dramatically different. This fact suggests that Foursquare contains a few items that are extremely popular, as it was already clear from Figure 2.

On the other hand, the RNN recommender obtained, with both datasets, comparable values of precision, but a very different coverage. For this reason, we can suppose that this approach, differently from the CRF recommender, is capable of better adapting itself to the characteristics of the dataset. The fact that we can draw such a conclusion supports the validity of Sequeval.

Furthermore, we have observed that the amount of time required to compute the evaluation metrics is linear with respect to the dataset size, if we do not consider the metric of diversity. In fact, the computational cost of the cosine similarity was too elevated when we analyzed the behavior of the random recommender with a more demanding dataset. However, because of the modular structure of `sequeval`, it is easy to avoid computing the metric of diversity for such a recommender.

In line with RQ3, we have described in Section 4.3 four different baseline recommenders. From the results of the experiments, it is possible to observe that the values obtained by some of them are fixed. For example, the MP recommender will always achieve a serendipity equal to 0, and a confidence equal to 1. Its perplexity is usually $+\infty$, if at least one of the recommended sequences is incorrect. The items suggested are, in fact, considered obvious by definition, and the algorithm is certain of the recommendation because its behavior is deterministic. In a similar way, the perplexity of the random recommender is equal to the total number of items available, i.e., $|\mathcal{I}|$, because of the definition of perplexity provided in Section 4.2.8.

These two baselines are methods commonly exploited in the literature for evaluating RSs. Additionally, we have proposed two techniques better suited for the sequence recommendation problem. The unigram recommender is similar to the MP one, but it is non-deterministic, and it obtained a higher novelty. In contrast, the bigram recommender is the most complex baseline, because it considers the previous item to suggest the next one. For this reason, it always achieved the lowest perplexity among the baselines considered.

6. Conclusions

In this paper, we have discussed the problem of recommending sequences of items tailored to the needs of a certain user. We have introduced an offline evaluation framework, called Sequeval, capable of handling this novel family of RSs in an offline scenario and we have developed an implementation of it that is publicly available on GitHub. We have included in such a framework an evaluation protocol and eight different metrics, to better capture the characteristics of the algorithms considered.

We have performed an empirical analysis of Sequeval by relying on it for conducting a comparison among four baselines, a CRF recommender, and an RNN-based one. The results have highlighted the fact that this framework is flexible, as it can be successfully applied in non-standard recommendation scenarios, such as with Yes.com, and complete, because of the different metrics included that consider several dimensions of the recommended sequences. In addition, we have observed that the RNN recommender system can effectively adapt itself to the characteristics of the training dataset. This conclusion supports the validity of Sequeval as a tool for conducting an offline experimentation.

Nevertheless, it is important being aware of the limitations of such a framework. The availability of many metrics may produce results which are difficult to interpret, especially if we are uncertain of what are the most relevant dimensions in our recommendation scenario. More in general, this is a common limitation of offline experiments, and it needs to be addressed by comparing the most promising algorithms in a subsequent online trail.

The formal definitions provided in Section 3 have been conceived as an extension of the seminal works on RSs capable of recommending sequences. For this reason, it is possible to set the length of the recommended sequences to 1 if we are interested in obtaining a single item. In a similar way, the item included in the seed rating can be exploited in order to set the context of the recommendation, but it can also be ignored if we want a sequence only based on the target user.

As future work, we plan to study in more depth what are the relationships among the different metrics included in the framework, with the purpose of integrating them in a final value that expresses the overall quality of the recommender. Such global score should be related to the recommendation scenario: for example, diversity may be important when recommending POIs to a tourist, but less useful in the musical domain.

Furthermore, it would be desirable to be able to create an evaluation framework that is adopted by a community of researcher when testing their algorithms, harmonizing the evaluation protocols and the interpretation of the performance of sequence-based RSs. For this reason, it is necessary to identify and to include in it some meaningful datasets, related to different domains that could be exploited during the evaluation phase, as well as other baselines and new RSs. The strategic choice of publicly releasing `sequeval` fosters further reuse and extension.

Finally, we would like to expand our evaluation framework to also support the online experimentation that should be performed after the offline analysis. The final goal of this work is, in fact, to enable researchers to spend more time in realizing the recommendation algorithm as they can rely on an evaluation framework that has already been designed and validated.

Author Contributions: conceptualization, D.M. and E.P.; methodology, G.R.; software, D.M.; validation, G.R. and M.M.; data curation, E.P.; writing—original draft preparation, D.M.; writing—review and editing, E.P., G.R. and M.M.; visualization, D.M.; supervision, M.M.

Funding: This work has been partially supported by the innovation activity PasTime (no. 17164) of EIT Digital.

Acknowledgments: Computational resources were provided by HPC@POLITO (http://hpc.polito.it).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ricci, F.; Rokach, L.; Shapira, B. Recommender Systems: Introduction and Challenges. In *Recommender Systems Handbook*, 2nd ed.; Springer: New York, NY, USA, 2015; Chapter 1, pp. 1–34.

2. Su, X.; Khoshgoftaar, T.M. A Survey of Collaborative Filtering Techniques. *Adv. Artif. Intell.* **2009**, 1–19. [CrossRef]
3. Balabanović, M.; Shoham, Y. Fab: Content-based, collaborative recommendation. *ACM Commun.* **1997**, *40*, 66–72. [CrossRef]
4. Basu, C.; Hirsh, H.; Cohen, W. Recommendation As Classification: Using Social and Content-based Information in Recommendation. In Proceedings of the Fifteenth National Conference on Artificial Intelligence, Madison, WI, USA, 26–30 July 1998; AAAI Press: Menlo Park, CA, USA, 1998; pp. 714–720.
5. Stai, E.; Kafetzoglou, S.; Tsiropoulou, E.E.; Papavassiliou, S. A holistic approach for personalization, relevance feedback & recommendation in enriched multimedia content. *Multimed. Tools Appl.* **2016**, *77*, 283–326. [CrossRef]
6. Campos, P.G.; Díez, F.; Cantador, I. Time-aware recommender systems: A comprehensive survey and analysis of existing evaluation protocols. *User Model. User-Adapt. Interact.* **2013**, *24*, 67–119. [CrossRef]
7. Ding, Y.; Li, X. Time weight collaborative filtering. In Proceedings of the 14th ACM International Conference on Information and Knowledge Management, Bremen, Germany, 31 October–5 November 2005; ACM Press: New York, NY, USA, 2005; pp. 485–492. [CrossRef]
8. Rendle, S.; Freudenthaler, C.; Schmidt-Thieme, L. Factorizing personalized Markov chains for next-basket recommendation. In Proceedings of the 19th International Conference on World Wide Web, Raleigh, NC, USA, 26–30 April 2010; ACM Press: New York, NY, USA, 2010; pp. 811–820. [CrossRef]
9. He, R.; Kang, W.C.; McAuley, J. Translation-based Recommendation. In Proceedings of the Eleventh ACM Conference on Recommender Systems, Como, Italy, 27–31 August 2017; ACM Press: New York, NY, USA, 2017; pp. 161–169. [CrossRef]
10. Wang, P.; Guo, J.; Lan, Y.; Xu, J.; Wan, S.; Cheng, X. Learning Hierarchical Representation Model for Next Basket Recommendation. In Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, 9–13 August 2015; ACM Press: New York, NY, USA, 2015; pp. 403–412. [CrossRef]
11. Zhou, B.; Hui, S.; Chang, K. An intelligent recommender system using sequential Web access patterns. In Proceedings of the IEEE Conference on Cybernetics and Intelligent Systems, Singapore, 1–3 December 2004; pp. 393–398. [CrossRef]
12. Quadrana, M.; Cremonesi, P.; Jannach, D. Sequence-Aware Recommender Systems. *ACM Comput. Surv.* **2018**, *51*, 1–36. [CrossRef]
13. Jurafsky, D.; Martin, J.H. *Speech and Language Processing*; Prentice Hall: Upper Saddle River, NJ, USA, 2008.
14. Herlocker, J.L.; Konstan, J.A.; Terveen, L.G.; Riedl, J.T. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.* **2004**, *22*, 5–53. [CrossRef]
15. Chen, S.; Moore, J.L.; Turnbull, D.; Joachims, T. Playlist prediction via metric embedding. In Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Beijing, China, 12–16 August 2012; ACM Press: New York, NY, USA, 2012; pp. 714–722. [CrossRef]
16. Feng, S.; Li, X.; Zeng, Y.; Cong, G.; Chee, Y.M.; Yuan, Q. Personalized Ranking Metric Embedding for Next New POI Recommendation. In Proceedings of the 24th International Conference on Artificial Intelligence, Buenos Aires, Argentina, 25–31 July 2015; AAAI Press: Menlo Park, CA, USA, 2015; pp. 2069–2075.
17. Ludewig, M.; Jannach, D. Evaluation of session-based recommendation algorithms. *User Model. User-Adapt. Interact.* **2018**, *28*, 331–390. [CrossRef]
18. Gunawardana, A.; Shani, G. Evaluating Recommender Systems. In *Recommender Systems Handbook*, 2nd ed.; Springer: New York, NY, USA, 2015; Chapter 8, pp. 265–308.
19. Jannach, D.; Lerche, L.; Kamehkhosh, I.; Jugovac, M. What recommenders recommend: An analysis of recommendation biases and possible countermeasures. *User Model. User-Adapt. Interact.* **2015**, *25*, 427–491. [CrossRef]
20. Costa, A.; Roda, F. Recommender systems by means of information retrieval. In Proceedings of the International Conference on Web Intelligence, Mining and Semantics, Sogndal, Norway, 25–27 May 2011; ACM Press: New York, NY, USA, 2011.
21. Pouli, V.; Kafetzoglou, S.; Tsiropoulou, E.E.; Dimitriou, A.; Papavassiliou, S. Personalized multimedia content retrieval through relevance feedback techniques for enhanced user experience. In Proceedings of the IEEE 13th International Conference on Telecommunications, Graz, Austria, 13–15 July 2015.

22. Albanese, M.; d'Acierno, A.; Moscato, V.; Persia, F.; Picariello, A. A Multimedia Semantic Recommender System for Cultural Heritage Applications. In Proceedings of the IEEE Fifth International Conference on Semantic Computing, Palo Alto, CA, USA, 18–21 September 2011.
23. Aggarwal, C.C. Mining Discrete Sequences. In *Data Mining*; Springer International Publishing: New York, NY, USA, 2015; Chapter 15, pp. 493–529.
24. Bellogín, A.; Sánchez, P. Collaborative filtering based on subsequence matching: A new approach. *Inf. Sci.* **2017**, *418–419*, 432–446. [CrossRef]
25. Hirschberg, D.S. A linear space algorithm for computing maximal common subsequences. *ACM Commun.* **1975**, *18*, 341–343. [CrossRef]
26. Albanese, M.; Chianese, A.; d'Acierno, A.; Moscato, V.; Picariello, A. A multimedia recommender integrating object features and user behavior. *Multimed. Tools Appl.* **2010**, *50*, 563–585. [CrossRef]
27. Albanese, M.; d'Acierno, A.; Moscato, V.; Persia, F.; Picariello, A. A Multimedia Recommender System. *ACM Trans. Internet Technol.* **2013**, *13*, 1–32. [CrossRef]
28. Amato, F.; Moscato, V.; Picariello, A.; Sperli, G. KIRA: A System for Knowledge-Based Access to Multimedia Art Collections. In Proceedings of the IEEE 11th International Conference on Semantic Computing, San Diego, CA, USA, 30 January–1 February 2017.
29. Su, X.; Sperli, G.; Moscato, V.; Picariello, A.; Esposito, C.; Choi, C. An Edge Intelligence Empowered Recommender System Enabling Cultural Heritage Applications. *IEEE Trans. Ind. Inform.* **2019**, 1. [CrossRef]
30. Bellogín, A.; Castells, P.; Cantador, I. Statistical biases in Information Retrieval metrics for recommender systems. *Inf. Retr. J.* **2017**, *20*, 606–634. [CrossRef]
31. Said, A.; Bellogín, A. Comparative recommender system evaluation. In Proceedings of the 8th ACM Conference on Recommender Systems, Foster City, CA, USA, 6–10 October 2014; ACM Press: New York, NY, USA, 2014; pp. 129–136. [CrossRef]
32. Turpin, A.H.; Hersh, W. Why batch and user evaluations do not give the same results. In Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, New Orleans, LA, USA, 9–12 September 2001; ACM Press: New York, NY, USA, 2001; pp. 225–231. [CrossRef]
33. Ge, M.; Delgado-Battenfeld, C.; Jannach, D. Beyond Accuracy: Evaluating Recommender Systems by Coverage and Serendipity. In Proceedings of the Fourth ACM Conference on Recommender Systems, Barcelona, Spain, 26–30 September 2010; ACM Press: New York, NY, USA, 2010; pp. 257–260. [CrossRef]
34. Paraschakis, D.; Nilsson, B.J.; Hollander, J. Comparative Evaluation of Top-N Recommenders in e-Commerce: An Industrial Perspective. In Proceedings of the IEEE 14th International Conference on Machine Learning and Applications, Miami, FL, USA, 9–11 December 2015; pp. 1024–1031. [CrossRef]
35. Bengio, Y.; Ducharme, R.; Vincent, P.; Jauvin, C. A Neural Probabilistic Language Model. *J. Mach. Learn. Res.* **2003**, *3*, 1137–1155.
36. Rijsbergen, C.J.V. *Information Retrieval*; Butterworth-Heinemann: Oxford, UK, 1979.
37. Sarwar, B.; Karypis, G.; Konstan, J.; Riedl, J. Analysis of recommendation algorithms for e-commerce. In Proceedings of the 2nd ACM Conference on Electronic Commerce, Minneapolis, MN, USA, 17–20 October 2000; ACM Press: New York, NY, USA, 2000; pp. 158–167. [CrossRef]
38. Yao, Y. Measuring retrieval effectiveness based on user preference of documents. *J. Am. Soc. Inf. Sci.* **1995**, *46*, 133–145. [CrossRef]
39. Järvelin, K.; Kekäläinen, J. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.* **2002**, *20*, 422–446. [CrossRef]
40. Ziegler, C.N.; McNee, S.M.; Konstan, J.A.; Lausen, G. Improving recommendation lists through topic diversification. In Proceedings of the 14th International Conference on World Wide Web, Chiba, Japan, 10–14 May 2005; ACM Press: New York, NY, USA, 2005; pp. 22–32. [CrossRef]
41. Noia, T.D.; Ostuni, V.C.; Rosati, J.; Tomeo, P.; Sciascio, E.D. An analysis of users' propensity toward diversity in recommendations. In Proceedings of the 8th ACM Conference on Recommender Systems, Foster City, CA, USA, 6–10 October 2014; ACM Press: New York, NY, USA, 2014; pp. 285–288. [CrossRef]
42. Vargas, S.; Castells, P. Rank and relevance in novelty and diversity metrics for recommender systems. In Proceedings of the Fifth ACM Conference on Recommender Systems, Chicago, IL, USA, 23–27 October 2011; ACM Press: New York, NY, USA, 2011; pp. 109–116. [CrossRef]

43. de Gemmis, M.; Lops, P.; Semeraro, G.; Musto, C. An investigation on the serendipity problem in recommender systems. *Inf. Process. Manag.* **2015**, *51*, 695–717. [CrossRef]
44. Herlocker, J.L.; Konstan, J.A.; Riedl, J. Explaining collaborative filtering recommendations. In Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work, Philadelphia, PA, USA, 2–6 December 2000; ACM Press: New York, NY, USA, 2000; pp. 241–250. [CrossRef]
45. Monti, D.; Palumbo, E.; Rizzo, G.; Morisio, M. Sequeval: A framework to assess and benchmark sequence-based recommender systems. In Proceedings of the Workshop on Offline Evaluation for Recommender Systems at the 12th ACM Conference on Recommender Systems, Vancouver, BC, Canada, 2 October 2018.
46. Chen, S.F.; Goodman, J. An empirical study of smoothing techniques for language modeling. *Comput. Speech Lang.* **1999**, *13*, 359–393. [CrossRef]
47. Sutton, C.; McCallum, A. An Introduction to Conditional Random Fields. *Found. Trends Mach. Learn.* **2011**, *4*, 267–373. [CrossRef]
48. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016. Available online: http://www.deeplearningbook.org (accessed on 8 May 2019).
49. Nocedal, J. Updating quasi-Newton matrices with limited storage. *Math. Comput.* **1980**, *35*, 773–782. [CrossRef]
50. Palumbo, E.; Rizzo, G.; Troncy, R.; Baralis, E. Predicting Your Next Stop-over from Location-based Social Network Data with Recurrent Neural Networks. In Proceedings of the 2nd Workshop on Recommenders in Tourism Co-Located with 11th ACM Conference on Recommender Systems, Como, Italy, 27 August 2017; pp. 1–8.
51. Sutskever, I.; Martens, J.; Hinton, G. Generating text with recurrent neural networks. In Proceedings of the 28th International Conference on Machine Learning, Bellevue, WA, USA, 28 June–2 July 2011; pp. 1017–1024.
52. Harper, F.M.; Konstan, J.A. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* **2015**, *5*, 1–19. [CrossRef]
53. Shannon, C.E. A Mathematical Theory of Communication. *Bell Syst. Tech. J.* **1948**, *27*, 379–423. [CrossRef]