Statistically Robust Evaluation of Stream-Based Recommender Systems

João Vinagre, Alípio Mário Jorge, Conceição Rocha, and Joao Gama

Abstract—Online incremental models for recommendation are nowadays pervasive in both the industry and the academia. However, there is not yet a standard evaluation methodology for the algorithms that maintain such models. Moreover, online evaluation methodologies available in the literature generally fall short on the statistical validation of results, since this validation is not trivially applicable to stream-based algorithms. We propose a k-fold validation framework for the pairwise comparison of recommendation algorithms that learn from user feedback streams, using prequential evaluation. Our proposal enables continuous statistical testing on adaptive-size sliding windows over the outcome of the prequential process, allowing practitioners and researchers to make decisions in real time based on solid statistical evidence. We present a set of experiments to gain insights on the sensitivity and robustness of two statistical tests—McNemar's and Wilcoxon signed rank—in a streaming data environment. Our results show that besides allowing a real-time, fine-grained online assessment, the online versions of the statistical tests are at least as robust as the batch versions, and definitely more robust than a simple prequential single-fold approach.

Index Terms—Recommender systems, data streams, evaluation

1 Introduction

In several fields of fundamental and applied research, online algorithms have been introduced to learn predictive and analytical models from continuous streams of data. In the case of recommender systems, online algorithms maintain recommendation models by updating them incrementally with data from a stream of user feedback. Evaluating such algorithms remains an open issue. The importance of making decisions based on solid evidence is undeniable not only in the industry, but also in the academic community, where robust, interpretable and repeatable evaluation methodologies are essential to conduct empirical studies. Bad decisions, supported by weak evidence – or no evidence at all –, potentially lead to loss of income or customer trust in the industry, and cause inefficiencies in the scientific discovery process.

Although there are several well studied methodologies to evaluate batch recommendation algorithms and static models [1], two important challenges are still present when evaluating incremental algorithms.

The first challenge is how to evaluate an ever-evolving recommender system. Static models can be easily evaluated by splitting available data in learning and testing subsets, training the model with the first and measuring the model's performance with the latter. Although it is possible to use the same methodology with incremental models in an offline setting, it is not trivial how to apply it in online environments, where the model is continuously updated with fresh data. In essence, batch evaluation protocols are designed for models learned offline and simply do not attempt to simulate the real-world environments in which incremental algorithms are designed to operate.

The second challenge is how to statistically validate hypotheses. In a batch environment, we can make statistically robust comparisons between several algorithms over several datasets using widely discussed methods [2], [3], [4]. In a data stream environment, the simple comparison between two algorithms over a single data stream remains insufficiently studied [5]. This happens for two reasons. First, performing multiple trials over a data stream is not a straightforward task, and therefore only instance-based tests such as McNemar are applicable [6]. Second, assuming that algorithms deal with an unbounded amount of data points, the scope of any assessment is obviously limited in time, which necessarily calls for protocols that implement frequent or continuous assessment. This paper provides a step forward in the evaluation of stream-based recommender systems, by applying Wilcoxon, an on-demand trial-based statistical test, over the outcome of any two alternative algorithms.

The application of statistical tests to stream-based recommender systems has been proposed in [7], [8], [9], using prequential evaluation. However, some questions remain unanswered:

1) When comparing the outcome of two alternative algorithms over the same data stream, can we observe the significance of the differences between them *at any desired point in time*?

- *J. Vinagre and A. M. Jorge are with the Laboratory of Artificial Intelligence and Decision Support (LIAAD), INESC TEC, 4200-465 Porto, Portugal, and also with the Faculty of Sciences, University of Porto, 4169-007 Porto, Portugal. E-mail: jnsilva@inesctec.pt, amjorge@fc.up.pt.*
- *C. Rocha is with the Centre of Power and Energy Systems (CPES), INESC TEC, 4200-465 Porto, Portugal, and also with the School of Management and Technology, Polytechnic of Porto, 4610-156 Felgueiras, Portugal. E-mail: conceicao.n.rocha@inesctec.pt.*
- *J. Gama is with the Laboratory of Artificial Intelligence and Decision Support (LIAAD), INESC TEC, 4200-465 Porto, Portugal, and also with the Faculty of Economics, University of Porto, 4200-464 Porto, Portugal. E-mail: jgama@fep.up.pt.*

Manuscript received 14 Nov. 2018; revised 15 Nov. 2019; accepted 12 Dec. 2019. Date of publication 17 Dec. 2019; date of current version 3 June 2021. (Corresponding author: João Vinagre.)
Recommended for acceptance by J. Caverlee.
Digital Object Identifier no. 10.1109/TKDE.2019.2960216

2) Can we use trial-based tests – such as Wilcoxon – in a streaming setting?
3) Assuming we are able to perform statistical tests online, how robust are those tests to Type I and Type II errors?

Question 1 has been partially answered in [7], using McNemar's test over a sliding window. However, there is no trivial method to set the window size, that is critical to the outcome of the statistical test. In this paper, we propose a methodology in which the window size is automatically adjusted online, enabling a parameter-free real-time monitoring of the statistical significance of differences between algorithms.

We also answer questions 2 and 3. Our starting point is the work on evaluation of classification methods over data streams by Bifet *et al.* [5], which we adapt to the recommendation problem and extend with the ability to provide continuous, online statistical validation.

We present the following contributions:

- We propose a framework to conduct anytime, on-demand statistical testing, with adaptive-size windows, using McNemar and Wilcoxon, two well-known non-parametric tests;
- We adapt and extend the three distributed k-fold validation methods proposed in [5], which are designed for classification problems, to recommendation problems, to enable the usage of trial-based tests in streaming environments;
- We provide insights on the sensitivity and robustness of the above methodologies, using two non-parametric statistical tests commonly used with machine learning algorithms;
- We illustrate the usefulness of our methodology by applying it to a scenario where we wish to decide, continuously and in real time, the best of two concurrent recommendation algorithms.

An always-available statistically robust assessment of algorithms is especially helpful in online evaluation methods that include user action [10], including A/B or multivariate testing [11]. Another advantage is the ability to automate many decisions based on the continuous comparison between algorithms. Potential applications include the automatic switching between algorithms, online parameter adjustment, or re-weighting of ensemble models. It can also help in the deployment of new algorithms, automatically activating them when the learning phase is stabilized. From a more analytical perspective, it allows us to gain insights on a long term evolution of the performance of recommendation models. We focus on the field of recommender systems, but we strongly believe that our findings can be generalized to other online learning tasks, such as online regression, reinforcement learning or semi-supervised learning.

In the remainder of the paper, we describe related work in Section 2, then we introduce the particular environment of online, stream-based recommender systems in Section 3, and prequential evaluation in Section 4, with emphasis in its application in the field of recommender systems. The framework to perform statistical tests with prequential evaluation in online streaming environments, is described in Section 5. Our experiments are described in Section 6, followed by a discussion with recommendations and limitations of this work in Section 7. In Section 8 we provide an illustrative application of our framework. Finally, we conclude in Section 9.

2 Related Work

Evaluation of online algorithms for classification problems is thoroughly studied in the field of data stream mining [12]. In [13], Gama *et al.* discuss a series of issues in the evaluation of stream-based classifiers. The authors illustrate the benefits of using prequential evaluation, by maintaining statistics of the outcome of the incremental learning process in sliding windows with arbitrary size, or using fading factors with fixed magnitude. More recently, Bifet *et al.* [5] propose a k-fold evaluation methodology, enabling statistical testing of classifiers that learn from data streams, inspired by the typical k-fold cross-validation of batch-learning algorithms. In the same paper, the authors also propose the usage of adaptive windows [14] in prequential evaluation, eliminating the need of arbitrarily setting the size for sliding windows.

In the field of recommender systems, several proposals have been made on incremental methods. Many of these proposals are based on classic train/test protocols designed for batch algorithms [15], [16], [17], [18], [19], [20], [21]. Other proposals evaluate algorithms using a chunk-based sequential approach [22] and [23]. The idea is to divide the dataset in N sequential chunks, using chunk n as training data and chunk $n+1$ for testing.

Prequential evaluation has been used in [8], [24], [25], [26], [27] in stream-based recommendation problems, but without statistical testing. In [7] the focus is on the actual prequential evaluation methodology, and proposes statistical testing, using McNemar's test over a sliding window with arbitrary size. Prequential evaluation is also used in a more elaborated protocol in [28], [29], [30], [31] with initial batch training, mainly to avoid cold-start issues. This protocol is divided in three stages. First, an initial model is learned in batch from the available data. Then, the model initiates incremental learning and uses the incoming data to evaluate the batch model. Finally, at the third stage, prequential evaluation is applied, with incoming data points being used for both learning and evaluation. Matuszyk *et al.* use statistical testing in [28] to validate results. The authors divide the dataset in non-overlapping subsets, and then apply the Friedman test [2]. Each subset is treated as an individual trial. This protocol enables straightforward application if trial-based statistical tests. However, although evaluation can be performed online – at least at the final stage –, statistical significance is only possible to asses offline, since it requires pre-processing a fixed size dataset.

In this paper we propose a methodology based on contributions by Bifet *et al.* [5] and Vinagre *et al.* [7], but extending them to the evaluation of Top-N recommender algorithms that learn from data streams. We additionally propose a methodology to enable on-demand statistical validation of results in real time by automatically maintaining variable-size windows with the outcome of the prequential process. This paper addresses the two challenges stated in Section 1, solving the limitations of previously contributed methodologies. We specifically focus on the limitations of [7] in the pairwise comparison of algorithms over a single data stream,

namely the uncertainty about statistical tests over a single instance of the problem, and the usage of data windows with arbitrary sizes.

3 Stream-based Recommender Systems

With the explosion of the volume, speed and variety of user-feedback data, recommendation with online streaming data has become an active field of study [32]. Stream-based algorithms make no assumptions on the order and rate of arrival of data. Runtime requirements are accounted for in three simple rules:

1) The data processing rate must be at least the same as the data arrival rate;
2) The amount of memory required by the algorithm must be independent of the amount of data points;
3) The algorithm must be able to learn from data in a single pass.

Many batch algorithms can be trivially used in a streaming environment as long as the above requirements are met. Perhaps the key difference between incremental and batch algorithms comes from rule 3: given that storage is not infinite, data will be archived, aggregated or discarded at some point, and therefore algorithms cannot assume past data is available.

Unlike batch approaches, stream-based recommendation does not have well-established and widely adopted methodologies to evaluate and compare algorithms. This raises two problems. First, comparison between contributions made by different authors is often misleading or even impossible. Second, it reduces reproducibility due to the added complexity of implementing the evaluation protocols. It also increases the chance of error and misinterpretation of results – either by over- or under-estimating them. In the following section, we briefly describe prequential evaluation, an evaluation method for algorithms that learn from data streams. We then argue in Section 4.1 that prequential evaluation can be effectively used in recommendation problems.

4 Prequential Evaluation

Stream-based algorithms can be evaluated using two alternative methods. The first alternative is to test the algorithm against a previously gathered *holdout* set, used exclusively for testing. This method is similar to batch evaluation methods, in the sense that an independent test set is used to measure the predictive ability of the model. The second method is *prequential* evaluation. This method does not require an independent test set. Instead, evaluation is performed over every data element in the data stream *before* it is used to update the model. In typical classification or regression problems, a model M is maintained over a stream D, consisting of vectors in the form $(\mathbf{x}, y)$, in which $\mathbf{x}$ is a set of features and y is the known label or target value. Prequential evaluation, using any given metric Δ goes through the following steps:

For every data element $(\mathbf{x}, y)$ in D:

1) Ask M for a prediction $\hat{y}$, given the feature set $\mathbf{x}$;
2) Score the prediction $\Delta(\hat{y}, y)$;
3) Use $(\mathbf{x}, y)$ to update the model;

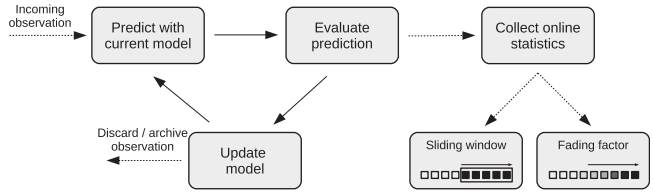

Fig. 1. Prequential evaluation.

Statistics over the score $\Delta(\hat{y}, y)$ can be maintained during the learning process. The prequential evaluation process is illustrated in Fig. 1.

Prequential evaluation has the advantage of not requiring an independent test set for testing, given that it performs testing on the actual stream. Furthermore, the prequential method has been shown to converge to the holdout evaluation method [13].

4.1 Using Prequential Evaluation in Recommendation Problems

Prequential evaluation for recommendation problems is made treating incoming user feedback data as a data stream [7], in the same test-then-learn scheme as depicted in Fig. 1: whenever a new user-item interaction arrives, the corresponding prediction is scored according to the actual observation. This new observation is then used to update the model.

In this paper, we focus on prequential evaluation with Top-N recommendation problems with implicit preference data. Each observation consists of a simple user-item pair (u, i) that indicates a positive interaction between user u and item i. For example, user u bought item i in an online store, user u "liked" a post i in a social network, or user u streamed track i in a music streaming service. The following steps are performed in the prequential evaluation process, for each new user-item pair (u, i):

1) If u is a known user, use the current model to recommend a list of items to u, otherwise go to step 3;
2) Score the recommendation list against the actual observed item i;
3) Update the model with (u, i) (optionally);
4) Proceed to – or wait for – the next observation

One convenient feature of prequential evaluation is that it is entirely applicable not only to incremental algorithms, but also to batch algorithms. This is the reason why step 3. is annotated as optional.

5 Statistical Tests Over Datastreams

Given two algorithms learning from the same data, the null hypothesis is that both have the same performance for a given metric. A Type I error occurs if the test wrongfully rejects the null hypothesis – i.e., if it detects a difference when there is none. A Type II error consists of failing to detect differences when they exist – i.e., to wrongfully fail to reject the null hypothesis.

There are two well known non-parametric tests widely used to compare two learning algorithms that learn from the same data: the McNemar's test and the Wilcoxon test.

McNemar's test is performed at the level of individual observations – no comparison between trials is involved – which makes it especially easy to use online with

stream-based algorithms. Given two alternative algorithms A and B, it works by keeping count of two quantities: the number of instances n_{10} for which the prediction of A is correct and the prediction of B is wrong, and the number of instances n_{01} for which the opposite occurs. These quantities are used to calculate the statistic:

$$M = \frac{(n_{10} - n_{01})^2}{n_{10} + n_{01}}, \quad (1)$$

M asymptotically follows a χ^2 distribution with one degree of freedom. For a significance level $\alpha = 0.01$, the critical value $M = 6.635$ is used. If $M > 6.635$ the null hypothesis is rejected. Given that McNemar is an instance-based test, the treatment of folds is done by concatenating them in the same order for the two alternatives. This is possible because the samples are perfectly paired and order is not relevant to calculate M. One problem with McNemar is that it fails to provide a reliable approximation if $n_{10} + n_{10} < 25$. Whenever this succeeds, the exact binomial test can be used instead of McNemar's. In the remainder of the paper, we refer to this conditional McNemar / Binomial test as McNemar only.

The Wilcoxon test uses the ranking of algorithms over several trials using different parts of the data for each trial. The test works by taking k trials and measuring the differences between the average score of the two algorithms A and B in each trial i: $x_{A,i} - x_{B,i}$, then ranking the absolute differences. For k trials, the statistic W is calculated based on the ranks R_i:

$$W = \sum_{i=1}^{k} [\text{sign}(x_{A,i} - x_{B,i}) \cdot R_i]. \quad (2)$$

For a significance level $\alpha = 0.01$ with $k = 10$, the critical value $W = 3$ is used. If $W > 3$, then a significant difference between the two algorithms is detected, and the null hypothesis is rejected.

Contrary to the McNemar's test, it is not trivial how to use the Wilcoxon test online over a single data stream, given that it requires multiple trials over different parts of the data.

5.1 Trial-based Statistical Tests Over Datastreams

The McNemar's test is convenient for data streams, since it has been shown to be effective without requiring multiple trials [6]. However this test is hardly considered robust. In fact, it is known to overestimate differences in many cases [5], incurring in Type I errors.

In an batch setting, the Wilcoxon test is currently considered the best alternative to assess the difference between two algorithms [2]. The problem with this approach is that it is not trivial to apply in stream-based scenarios. In [5], three alternative methods are proposed to perform K-fold validation with classifiers that learn from data streams. The main idea is to distribute data points across k versions of the classifier in one of the following ways:

- k-fold split-validation: each data example is used for training in $k - 1$ classifiers and testing in the remaining classifier;
- k-fold bootstrap valuation: each example is assigned for training to each classifier according to a $Poisson(1)$ distribution, and used for testing in the remaining classifiers;
- k-fold cross-validation: each example is used for training in 1 classifier and for testing in the other $k - 1$ classifiers.

The prequential version of the above methods simply uses all training examples also for testing, using the prequential workflow. This translates into a setting in which every example is tested in all k versions of the classifier, and only used for training in some of them, according to one of the three distribution strategies. A statistical test can be used in conjunction with the distributed k-fold validation methods, using each fold as a trial.

5.2 Continuous Statistical Testing

In [13], Gama *et al.* have illustrated the usage of statistical tests over data streams using McNemar's test, using a sliding window over the outcome. This technique has also been used in [7] for recommendation problems, partially answering our first research question. However, the size of the sliding window is a user-given parameter that has a critical impact on the outcome, and one that is not trivial to set. Very large windows tend to underestimate differences, whereas very short ones potentially overestimate them. To fully answer research question 1, we need an method that *automatically* handles the window size.

To solve this problem, we propose using ADWIN [14], a data-based mechanism to automatically resize a sliding window over a data stream. We use this mechanism to maintain a window that we can use to perform statistical tests any point in time. In [5], Bifet *et al.* propose the usage of ADWIN to monitor performance metrics in a prequential evaluation process, but in a single fold, and they do not perform continuous statistical testing.

Given a pair of stream-based recommender algorithms, we measure their performance separately, using a common metric. For example, we may have two alternative algorithms A and B running in parallel, from which we take a sequence of measurements using a specific metric. The measurements from both algorithms form a pair of sequences, consisting of individual scores obtained in the prequential process. We wish to be able to automatically perform a statistical test to assess the significance of the differences between A and B at any point of the learning process. To do this, we use the most recent pair of windows, containing the outcomes of both recommenders. Naturally, these sequences have different sizes – if they are different. The problem here, is that to use statistical tests that require paired samples, we need sequences with the same size, for both recommenders, in which case we have four choices:

- Compute window sizes separately and use the shortest one;
- Compute window sizes separately and use the largest one;
- Compute window sizes separately and use a window with the average size;
- Compute a window size on a third sequence obtained by combining the two original sequences.

We use the first option, that is to compute window sizes separately for both recommenders and use the shortest one. The idea is that we wish to detect changes in the performance of each algorithm relative to the other. ADWIN reduces the

window size when change is detected in the distribution of the sequence. This means that the fastest changing sequence will have a shorter window. Choosing the largest window for both sequences, or even an average size window, would underestimate the changes of the fastest changing algorithm – the one with the shortest window – because a too large, non-representative window would be used. We argue that the risk of over-estimating fluctuations in the recommender with the largest window as a result of artificially shortening it is already mitigated by the fact that ADWIN has not triggered the shortening of the window in the first place, which is a strong indication that it is within a stationary interval.

5.3 Dealing with Independence

Given the data dependence problem in recommendation, we propose user-based distribution of data points for k-fold distributed validation, using the same three methods proposed in [5]. By distributing users, rather than individual observations, across folds, we greatly reduce the dependence between folds. While we know that total independence does not exist between users, given the effects of local, gloabal and item-level changes, the independence assumption becomes much safer nevertheless.

User-based distribution is only possible in the prequential version of the distributed k-fold validation methods described in Section 5.1, because all user-item pairs (u, i) can be evaluated in the folds to which user u was assigned. In the original version of the validation methods, any data point appearing in the stream would have to be used either for training or for testing, but never for both. This would be impossible in a user-based split of the data points because the standard usage-based recommendation model is unable to make predictions for unknown users – a well known problem in all recommendation tasks, known as *cold-start*.

We revisit the three k-fold validation methods described in Section 5.1, adapting them to the recommendation problem. The k-fold distribution methods basically consist of building k models in k user-based samples of the original stream, with the following characteristics:

- k-fold split validation: each fold has a model built in completely independent user sets;
- k-fold bootstrap validation: each fold has a model built over a bootstrap sample of the users from the original stream;
- k-fold cross-validation: each fold has a model built in a stream containing roughly $\frac{k-1}{k}$ users from the original stream.

The distribution can easily be performed online, using a $u \rightarrow k$ mapping matrix. For every new user, we randomly select the fold(s) to which she is assigned, according to one of the above three methods.

As in [5], the only alternative in which there is no overlap between folds is split validation, however each fold is learned over a potentially small proportion of the data, which increases sparsity in the inverse proportion. High sparsity is known to be problematic in recommendation problems [33]. Additionally, online learning algorithms may have different convergence speeds, which potentially biases evaluation, favoring fast-converging algorithms if not enough data is available. On the other extreme, cross-validation trains each fold on a potentially high proportion of the data, which alleviates the sparsity and convergence speed problems, at the cost of increased computation. However, it also causes folds to be trained in largely overlapping subsets of the data, for which assumptions of independence cannot be held.

5.4 Trial-Based Tests for Stream-based Recommenders

To address our second research question, we apply the three data-distribution strategies proposed in [5]. However, the direct application to recommender systems is not straightforward. All three methods presented in Section 5.1 are designed and studied for classification problems, for which several assumptions about data are relatively safe to make. One assumption in classic supervised learning is that the probability distributions of class labels or target values are stationary, *at least most of the time*. When a concept drift occurs – i.e., there is a change on the probability distribution –, detection mechanisms can be used to adapt the model. In a stationary data stream – or within a stationary interval –, individual observations are not inter-correlated. In other words, the sequence of events is not important. This allows us to randomly distribute data across instances of the algorithm – folds – without hurting the assumption that each fold is an independent instance of the problem, and thus enabling the usage of a trial-based statistical test of our choice, such as is done in [5]. However, the same assumptions cannot be safely made about user feedback data. One basic principle of recommender systems is that observations taken from the same user are highly correlated. Furthermore, several works surveyed in [34] and [32] actually exploit time or sequence dependence to improve recommendation accuracy, which proves that dependence between observations exists – otherwise it would not be exploitable. Moreover, drift detection mechanisms for classification or regression cannot be trivially applied to recommendation problems, given that user preferences may change independently of each other. At most, we can use them to detect global changes, which may be useful, but do not detect changes of individual users.

6 Experiments and Validation

To assess the robustness of our methodology, we conduct two sets of experiments. In the first set of experiments, we wish to evaluate how robust are the statistical tests to changes in the initialization of an algorithms that are not expected to induce significant changes in their outcome. This set of experiments provides insights regarding the propensity of statistical tests to Type I errors. In the second set of experiments, we measure the sensitivity of the tests to controlled perturbations in the outcome of algorithms. This set of experiments allows us to verify which test signals significant changes faster, providing valuable clues on the propensity to Type II errors.

We use the same framework to compare rejection rates between offline tests – as would be done in a batch scenario – and ADWIN tests. The difference is that the first goes through the complete prequential process for the entire dataset and performs the statistical test at the end, while with the latter we continuously perform statistical tests *during* prequential evaluation, effectively simulating a real-world

TABLE 1
Dataset Description

Dataset	Application	Events	Users	Items
PLC-PL	Music playlisting	111 942	10 392	26 117
PLC-STR	Music streaming	588 851	7 580	30 092
ML1M	Movie rating	226 310	6 014	3 232
YMUSIC	Music rating	152 183	2 000	60 215

online environment. Although feasible, for the sake of computational resources, we do not perform statistical tests with each new observation. Instead, we issue statistical tests at regular intervals, at every Nth example in the stream. In all our experiments we use $N = 100$. Note that the actual number of statistical tests performed is the length of the data stream (so far) divided by N, whereas a single statistical test is performed in the offline case.

We use McNemar's test and the Wilcoxon signed-rank test to measure rejection rates. Both are non-parametric statistical tests, but only Wilcoxon is a trial-based test. In order to use the McNemar's test, we simply concatenate the 10 folds for each trial – using the same order – and run the test over these two concatenated sequences.

6.1 Algorithm and Metric

To assess the robustness to changes in the random seed, we need an incremental recommendation algorithm that has a random initialization, allowing us to run two different, but hypothetically equivalent versions. We use ISGD [24], since it is the simplest algorithm that fulfills the requirement.

We measure top-N accuracy using the Hit Ratio at cutoff 20 – HR@20. This metric is measured in the prequential evaluation process. At the prediction step for pair (u, i), we recommend 20 items to user u and score the prediction with HR@20 = 1 if the item i is within the 20 recommended items, and HR@20 = 0 otherwise. This produces a binary sequence of 0s and 1s, corresponding to the scores obtained for each observation. The exception is when any user u first occurs – a cold-start situation. It is not trivial to make a recommendation to a user that is not yet modeled. Given that in this paper we are not interested in cold-start problems, for such cases, we bypass the prediction and evaluation step and the user-item pair (u, i) is only used for training.

6.2 Datasets

We use four publicly available datasets, described in Table 1. PLC-PL[1] is a dataset gathered from a music social network, with music playlisting activity. Each tuple (u, i, t) consists of a user u adding a music track i to her personal playlist, at timestamp t. The dataset is ordered by timestamp, allowing us to feed a stream of (u, i) pairs to an incremental algorithm. PLC-STR was collected from the same source as PLC-PL, however, it consists of a music streaming log. ML1M is a binary version of the Movielens-1M movie ratings dataset[2], in which we keep only the rating 5 – in a 0 to 5 scale. Given that the ratings are timestamped, we are also able to produce a stream of pairs (u, i) from that dataset. YMUSIC is a subset of the Yahoo! Music dataset[3], which originally contains ratings given by users to music tracks in a 0 to 100 scale. We first filter out ratings lower than 90 and then randomly sample 2000 users.

TABLE 2
Null Hypothesis Rejection Rate for Same Algorithm With Different Initialization Seed, Over All Datasets

Method	Wilcoxon	W. ADWIN	McNemar	M. ADWIN
One-fold	-	-	0.121	0.097
Split	0.004	0.010	0.044	0.021
Bootstrap	0.003	0.007	0.088	0.070
Crossval	0.002	0.016	0.150	0.144

6.3 Robustness to Type-I Errors

In the first set of experiments, we follow the same strategy of [5] to measure the robustness of the evaluation methodology to slight changes, with an incremental Top-N recommendation algorithm that has random initialization. We run the algorithm 100 times with different random seeds, with four publicly available datasets, for each of the three k-fold distributed validation strategies – split, bootstrap, and cross-validation – with $k = 10$, in a total of three-hundred ten-fold runs for each dataset. We also run the the algorithm 100 times using a single fold, for reference. We then perform statistical tests over distinct 50 pairs of runs, and measure the significance of the differences between the two runs of each pair. Then we count how many times the null hypothesis is rejected. We formulate the null hypothesis that two versions of the same algorithm, using the same data and the same hyperparameters, have the same accuracy, independently of the random seed.

Here, we make the assumption that changing the random seed does not induce significant changes in the algorithm's outcome. In principle, random initialization should not introduce significant changes, otherwise it should be treated as a hyperparameter. Under this assumption, the proportion of null hypothesis rejections should not be higher than the significance level 0.01.

Table 2 contains the null hypothesis rejection rate over all four datasets – rates are the average of datasets, weighted by dataset length –, using the batch and prequential versions of Wilcoxon and McNemar. Each row in the table corresponds to a validation method. The first line corresponds to one-fold test, which is only compatible with McNemar. The ADWIN version of the one-fold test is similar to the statistical testing framework used in [7], except that here we are using adaptive windows, while in [7], arbitrary fixed-size windows are employed. Note that for the offline versions, a single test is made for each of the 50 runs, whereas for the ADWIN version, a test is performed every 100 examples, in each of the 50 runs. Naturally, the number of tests is much higher with the ADWIN version.

The differences between Wilcoxon and McNemar are depicted in Fig. 2. The gray line marks the 0.01 significance level. We can make three observations. First, it is obvious that McNemar triggers null hypothesis rejections much more often than Wilcoxon, regardless of if we are using the

1. https://rdm.inesctec.pt/dataset/cs-2017-003- playlisted_tracks.tsv
2. https://grouplens.org/datasets/movielens/
3. https://webscope.sandbox.yahoo.com/catalog.php?datatype=r - Dataset R2

Fig. 2. Rejection rate over all datasets with varying random seed. Note that the scale is truncated to 0.15 rejection rate, to facilitate the visualization of rates close to the significance value of 0.01.

offline test, or the online ADWIN test. While the Wilcoxon test remains below the significance level of 0.01 in most cases, McNemar largely overshoots it in most cases. Second, online tests tend to have a slightly higher rejection rate with Wilcoxon and slightly lower rejection rate with McNemar. However, the difference between offline and online tests are not large. Third, McNemar is much more sensitive to the data distribution strategy. For example, the rejection rate of the online ADWIN version of McNemar varies from 0.021 with split validation, to 0.144 with cross-validation, while Wilcoxon yields 0.010 to 0.016 respectively, much closer to the expected significance value of 0.01. Both offline and online Wilcoxon obtain very similar rejection rates with all three validation methods.

We break down these results by dataset in Table 3. With Wilcoxon, rejection rates appear to be consistent across all datasets, whereas McNemar has higher variability – especially noticeable with ML1M.

6.4 Robustness to Type II Errors

To assess the robustness to type II errors, we are interested in detecting how often is the null hypothesis wrongfully retained. To do that, we follow a slightly different strategy from [5]. Bifet *et al.* introduce noise in the predictions of one version of the algorithm and test it against a clean version. Our strategy is to artificially improve one of the sequences of prequential scores, and test it against the non-modified version, changing the scores rather than changing the actual recommendations. We do this because we are dealing with a top-N recommendation problem, in which randomly changing recommended items has an unpredictable impact on the metric. For instance, in a recommendation list of 10 items, changing one random item in that list during the prequential process would most of the time have little or no impact in most top-N metrics. By directly changing the outcome, we have a measurable difference between runs. We take the score sequences from 50 runs – initialized with different random seeds – and then duplicate them and artificially improve the duplicated versions. Then we test each one of the original outcomes against its "improved" version. If this change is significant, the statistical test should be able to detect a difference between the two runs and reject the null hypothesis. Note that tests applied here are one-sided, given that we know that the induced differences are always in the same direction.

We measure how often the null hypothesis is retained under 3 different levels of positive perturbations. Our approach is to test the original output of ISGD against an artificially improved version of itself. In our setting, the outcome of the prequential process is a binary sequence of 0's and 1's that are the HR@20 measured at each step. We change this binary sequence, randomly replacing 0s with 1s, with three different probabilities $\delta \in \{0.0001, 0.0005, 0.001\}$. We then compare the original and the modified versions under the null hypothesis that there is no significant difference between them. Although we cannot definitely read the rejections as Type II errors, we can use them to understand the behavior of statistical tests and between the three data distribution methods – split versus bootstrap versus cross-validation.

In Table 4, we summarize the results of experiments similar to the ones in Section 6.3, but using this strategy. For the

TABLE 3
Null Hypothesis Rejection Rate for Same Algorithm with Different Initialization Seed, Broken Down by Dataset

Dataset	Wilcoxon	Wilcoxon ADWIN	McNemar	McNemar ADWIN
PLC-PL one-fold	-	-	0.020	0.128
PLC-PL split	0	0.017	0.080	0.027
PLC-PL bootstrap	0	0.005	0.180	0.056
PLC-PL crossval	0.020	0.025	0.160	0.145
PLC-STR one-fold	-	-	0.120	0.103
PLC-STR split	0	0.007	0.040	0.018
PLC-STR bootstrap	0	0.002	0.080	0.070
PLC-PL crossval	0	0.020	0.200	0.186
ML1M one-fold	-	-	0.020	0.014
ML1M split	0.020	0.010	0.040	0.011
ML1M bootstrap	0	0.018	0	0.020
ML1M crossval	0	0.002	0.020	0.020
YMUSIC one-fold	-	-	0.220	0.171
YMUSIC split	0	0.016	0.040	0.045
YMUSIC bootstrap	0.020	0.012	0.180	0.156
YMUSIC crossval	0	0.011	0.140	0.168

TABLE 4
Null Hypothesis Rejection Rate for Same Algorithm With Noise Filter p, Over All Datasets

Method	Wilcoxon	Wilcoxon ADWIN			McNemar		McNemar ADWIN		
δ	all	.0001	.0005	.001	.0001	others	.0001	.0005	.001
One-fold	-	-	-	-	0.896	1	0.214	0.817	0.912
Split	1	0.073	0.424	0.589	1		0.168	0.801	0.920
Bootstrap	1	0.631	0.870	0.927	1		0.857	0.980	0.990
Crossval	1	0.679	0.873	0.908	1		0.869	0.987	0.995

sake of space, we aggregate all three values of p in the same column for the batch versions, since all yield the same result – except the one-fold version, as noted. We also present plots for the same results in Figs. 3, 4, and 5. In these plots, we omit the batch versions of the tests. Note that here, if we assume that the null hypothesis is wrong, the rejection rate should be close to 1.

The first observation is that the offline validation methods almost always reject the null hypothesis, independently of the algorithm improvement level δ. The only exception is the offline McNemar test with one-fold which yields a lower rejection rate. The second observation is that naturally, rejection rates increase as we increase δ, regardless of the test. Another immediate observation is that McNemar has consistently higher rejection rates. In this set of experiments, rejection rates are more sensitive to the validation method. Split validation as a much lower rejection rate for all three levels of δ.

We also present results broken down by dataset in Table 5 for reference.

7 Discussion

In Section 6.3, our main objective is to assess how consistent the statistical test is with a reasonable assumption that the null hypothesis is true. There is still a chance that the same algorithm with different initialization can actually yield significantly different results. However, we know that in at least *most* of the 50 runs, with different initialization for both versions of the algorithm, changing the random seed should not trigger the rejection of the null hypothesis. From the practitioner's point of view, changing the random seed should originate slightly different, but in any case equivalent models, at least in terms of accuracy.

We know that an overly sensitive statistical test will be more likely to yield Type I errors. Conversely, we know that a more conservative statistical test will be more likely to yield Type II errors. Carefully looking at results of Tables 2 and 4 together, we are able to observe that McNemar is generally more sensitive to changes. However it is also more unstable with respect to both data and validation method. When changing the random seed of the algorithm, McNemar has an inconsistent behavior regarding the dataset, with a much lower rejection rate with ML1M than with other datasets. This is especially true for the offline version of the tests. Online statistical testing with Wilcoxon correctly fails to reject the null hypothesis more often than McNemar – see Table 2. At the same time, the differences between McNemar and Wilcoxon are smaller when we expect to see high rejection rates – see Table 4. In short, Wilcoxon is almost as sensitive as McNemar, but at the same time it is less likely to incur in Type I errors.

Another dimension is the data distribution technique. McNemar tends to trigger the rejection of the null hypothesis more often as we increase the number of available points. With split-validation, McNemar has lower rejection rate, followed by bootstrap-validation. With cross-validation, the rejection rate is even higher. Wilcoxon, on the other hand, is more stable, although we see in Section 6.4 that its rejection rate is also lower with split-validation.

Casting multiple instances of the problem involves a considerable computational overhead in most cases. From this perspective, the strategy that requires less computational resources is the split-validation method. In this case the computational effort is practically the same as running a single instance of the problem with all the users in the data stream. This is also the method that guarantees more independence between folds, since every user is only present in a single fold. The disadvantage of split validation is that each recommendation model is built using only a small fraction of the data – more specifically, $1/k^{th}$ of the data with k folds.

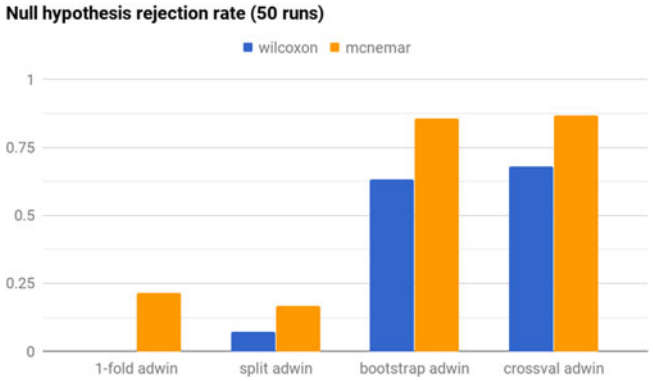

Fig. 3. Rejection rate over all datasets with noise filter $\delta = 0.0001$.

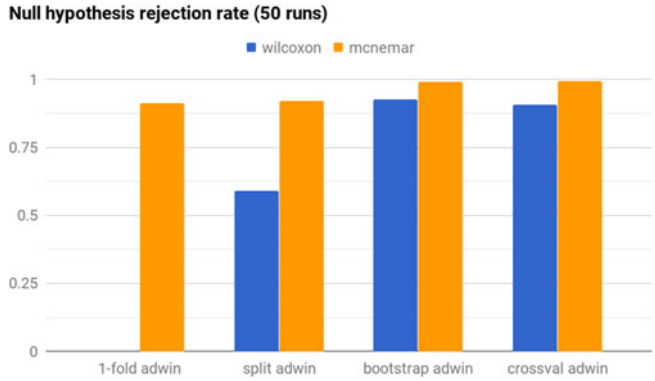

Fig. 4. Rejection rate over all datasets with noise filter $\delta = 0.0005$.

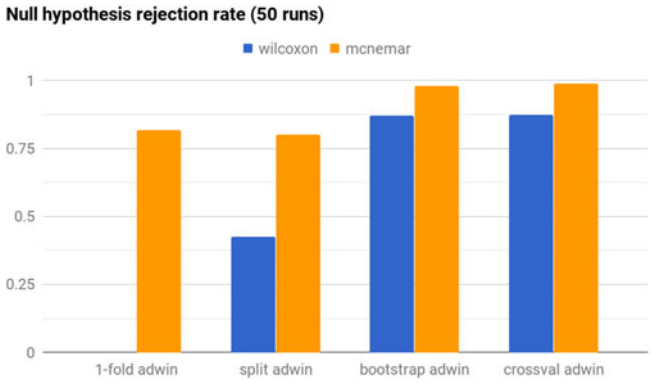

Fig. 5. Rejection rate over all datasets with noise filter $\delta = 0.001$.

This increases sparsity by the same factor. Recommendation algorithms are sensitive to sparsity and tend to degrade as sparsity increases. Moreover, split-validation increases by a factor of k the number of necessary examples for tested algorithms to converge to their optima.

On the other extreme, each instance in cross-validation uses most of the data available – $k - 1/k$ examples with k folds. This makes each instance of the problem very similar to the outcome of the algorithm if trained with all the users in the data stream, which is the real-world scenario. However, there is a big overlap between folds, which raises questions about the independence between folds. Cross-validation is also the most resource-demanding option, with a computational overhead many times higher than a single instance of the problem with the original data stream.

Bootstrap validation can be seen as a trade-off between the problems described above – computational overhead, data requirements and independence. Most of the users in the data stream are modeled in all instances, but there some separation is provided by bootstrap sampling. Computational effort is also reduced in relation to cross-validation.

TABLE 6
Test Robustness to Type I and Type II Errors

Method	Wilcoxon ADWIN		McNemar ADWIN	
Error type	Type I	Type II	Type I	Type II
One-fold	-	-	ø	ø
Split	✓	ø	∼	∼
Bootstrap	✓	✓	ø	✓
Crossval	✓	✓	ø	✓

Methods are marked as robust (✓), moderately robust (∼) and not robust (ø).

7.1 Recommendations

Taking our experimental results into account, we devise a summary of recommendations in Table 6. This table indicates which variants of the methodology are robust to Type I and Type II errors, using either Wilcoxon or McNemar.

We recommend the bootstrap method with the Wilcoxon signed-rank test as a rule of thumb. However we stress that this choice is dependent on the practical problem in hands, and necessarily needs to consider several factors, such as data availability, available resources and the overall objective of the online statistical tests. These can be used for a variety of tasks, such as automatic hyperparameter tuning, changing weights of individual algorithms in ensembles, simple swapping between algorithms, or online tests – e.g., A/B or multivariate tests – involving measurements of user activity. All these examples naturally have different requirements and constraints.

We point out that the main contribution of this paper is a framework that enables the usage of trial-based statistical tests on stream-based recommendation problems. The comparison between McNemar and Wilcoxon in the context of this paper does not replace an in-depth study about the robustness of the statistical tests themselves and therefore should not be generalized to other settings or tasks.

TABLE 5
Null Hypothesis Rejection Rate for Same Algorithm With Noise Filter p, Broken Down by Dataset

Dataset	Wilcoxon	Wilcoxon ADWIN			McNemar		McNemar ADWIN		
δ	all	.0001	.0005	.001	.0001	others	.0001	.0005	.001
PLC-PL 1-fold	-	-	-	-	0	1	0	0.757	0.935
PLC-PL split	1	0.026	0.398	0.756	1		0.082	0.805	0.937
PLC-PL bootstrap	1	0.555	0.885	0.954	1		0.820	0.965	0.988
PLC-PL crossval	1	0.625	0.910	0.974	1		0.854	0.975	0.993
PLC-STR 1-fold	-	-	-	-	1		0.115	0.778	0.898
PLC-STR split	1	0.002	0.267	0.442	1		0.002	0.736	0.902
PLC-STR bootstrap	1	0.569	0.822	0.900	1		0.830	0.981	0.991
PLC-STR crossval	1	0.591	0.811	0.854	1		0.828	0.988	0.996
ML1M 1-fold	-	-	-	-	1		0.551	0.915	0.957
ML1M split	1	0.267	0.724	0.828	1		0.559	0.918	0.955
ML1M bootstrap	1	0.805	0.951	0.975	1		0.925	0.985	0.993
ML1M crossval	1	0.871	0.966	0.982	1		0.954	0.990	0.995
YMUSIC 1-fold	-	-	-	-	1		0.254	0.865	0.935
YMUSIC split	1	0.091	0.607	0.756	1		0.292	0.876	0.937
YMUSIC bootstrap	1	0.672	0.924	0.954	1		0.888	0.977	0.988
YMUSIC crossval	1	0.775	0.949	0.974	1		0.915	0.985	0.993

7.2 Limitations

The methodology proposed in this paper enables the on-demand assessment of which one of two concurrent recommendation algorithms is better. As we have shown, it can be used with recommenders running in production settings. However, its application scope is naturally limited to the setting with two algorithms that run over a single user feedback stream. The Friedman test is typically used in batch learning to rank multiple algorithms for a given dataset. To perform this evaluation over multiple datasets, Demsăr *et al*. recommend using a post-hoc test [2]. However, the possibility to apply the Friedman test online to stream-based recommenders remains an open problem.

From an operational perspective, distributing data across several running nodes obviously has computational costs. However, given that stream-based algorithms have a single core sequential processing nature, added that our proposal is trivially parallelizable, this overhead can be easily handled by standard multicore processing units in real time.

8 Case Study: Dynamically Selecting the Best of Two Algorithms

In this section, we provide an example application of our framework. We perform continuous online testing to two concurrent algorithms that learn from the same data stream. The task is to perform continuous statistical tests to automatically determine which is the best algorithm at any point during the prequential process.

8.1 Experimental Setup

To illustrate how one of two competitor algorithms A and B may be ideal at different stages, we need to setup an experiment where we know beforehand that a shift between the performance of the two algorithms occurs, and when it occurs. The statistical test(s) should be able to detect a statistically significant improvement of one algorithm with respect to the other close to the region where the change occurs. To enable such observation, our strategy is to use two datasets X and Y with the two algorithms A and B. Datasets X and Y are chosen with the prior knowledge that algorithm A has higher accuracy than algorithm B with dataset X and that the opposite occurs with dataset Y. If we simply concatenate both datasets and compare algorithms A and B with our online evaluation framework on the concatenated dataset, we should be able to detect the change with statistical guarantees provided by the tests.

Using this strategy, we compare ISGD against an incremental version of a popularity-based algorithm (MostPopular) that simply recommends the most popular items. This algorithm has no personalization whatsoever. Recommendations consist of a list of the top-N most frequently seen items, regardless of the user.

8.2 Dataset

The simplicity of the MostPopular algorithm allows us to easily synthesize a dataset for which the algorithm achieves optimal performance. We synthesize a dataset composed of 5000 virtual users and 10 virtual items. All possible (u, i) pairs occur exactly once in the dataset, in randomized order, leading to 50.000 interaction pairs. This means that all users see the same 10 items. In this dataset, the MostPopular algorithm should obviously have fast convergence, since it only needs to recommend the same 10 items to all users.

Next, we concatenate PLC-STR to this synthesized dataset. As a consequence, the resulting dataset contains an abrupt shift after its 50.000$^{\text{th}}$ point, where the transition between the synthesized dataset and PLC-STR occurs. Naturally, ISGD has a clear advantage with PLC-PL, given its ability to deal with more complex problems.

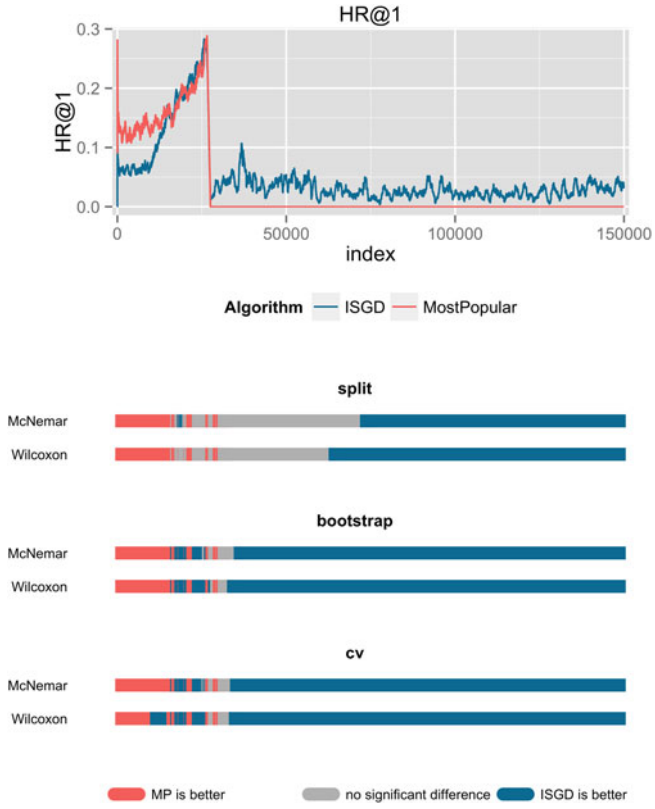

Fig. 6. Evolution of HR@1 using a moving average with $n = 1000$ (top plot). The bottom three plots depict the outcome of the continuous statistical tests over the sequence depicted on the top plot, using the three variants of the methodology – split-, bootstrap- and cross-validation.

8.3 Experiment Protocol

We apply the methodology proposed in Sections 5.1 and measure the results using Hit Ratio at cutoff 1 – HR@1 – for both ISGD and MostPopular (MP). We perform continuous statistical tests over the stream of results to assess the significance of the differences between the two outcomes, using all three distributed validation methods – *split* validation, *bootstrap* validation and *cross*-validation. We then plot both the outcome of HR@1 for both algorithms and the outcome of the statistical tests in Fig. 6.

8.4 Results

The top plot in Fig. 6 depicts the evolution of HR@1 for ISGD and MostPopular (MP). The shift between the two original datasets – i.e., the point where they were concatenated – predictably shows a steep degradation of both algorithms. After this point, ISGD is able to recover closely after, given its

ability to personalize recommendations, while MP drops to approximately zero, and never recovers. The three bottom plots in Fig. 6 depict the evolution of two statistical tests – McNemar and Wilcoxon – corresponding to the same experiment, and using the three validation variants studied in this paper. The three plots are intentionally aligned with the top plot to facilitate the correspondence between both types of illustration. Using split validation, we observe relatively few fluctuations of the significant tests. There is also a clearly longer delay until the statistical tests detect a rather obvious superiority of ISGD after the dataset shift. Bootstrap validation signals more fluctuations between the relative performance of algorithms, and reacts much faster to the dataset shift. Finally, cross-validation with McNemar is very similar to the same test using bootstrap, except that it reacts slightly faster to the shift. The Wilcoxon test shows more fluctuations, even those that are not easily seen in the HR@1 plot.

In general, all tests and testing variants are consistent with the actual results in terms of Hit Ratio, which is not surprising. However, we see that we are able to react to changes faster using the Wilcoxon test. This is consistent with the experimental work shown in Section 6.

9 Conclusions and Future Work

Reliable evaluation of online, incremental recommendation algorithms is an important issue for both the academia and industry. In this paper, we propose a methodology to evaluate incremental recommender systems with the ability to compare algorithms online, in real-time, with statistically validated results. To our current knowledge, this is the first systematic and comprehensive work that organizes a robust framework to conduct experiments online, using prequential evaluation, and at the same time statistically validate results. We conduct a series of experiments to assess the robustness of online statistical tests over automatically adaptive-size windows over the prequential outcome in any given metric. Our results suggest that using a bootstrap-based splitting of users online across several folds is beneficial, especially using the Wilcoxon signed-rank test. Future work includes studying the behaviour of statistical tests with non-binary metrics, more specifically ranking metrics. Special attention should also be given to highly dynamic domains such as news and Point-Of-Interest recommendation.

Acknowledgments

This work was financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia within Project: UID/EEA/50014/2019.

References

[1] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 5–53, 2004.

[2] J. Demsar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006.

[3] S. Garcia and F. Herrera, "An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons," *J. Mach. Learn. Res.*, vol. 9, pp. 2677–2694, 2008.

[4] A. Benavoli, G. Corani, and F. Mangili, "Should we really use post-hoc tests based on mean-ranks?" *J. Mach. Learn. Res.*, vol. 17, pp. 5:1–5:10, 2016.

[5] A. Bifet, G. D. F. Morales, J. Read, G. Holmes, and B. Pfahringer, "Efficient online evaluation of big data stream classifiers," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 59–68.

[6] T. G. Dietterich, "Approximate statistical test for comparing supervised classification learning algorithms," *Neural Comput.*, vol. 10, no. 7, pp. 1895–1923, 1998.

[7] J. Vinagre, A. M. Jorge, and J. Gama, "Evaluation of recommender systems in streaming environments," in *Proc. Workshop Recommender Syst. Eval.: Dimensions Design Conjunction*, 2014, pp. 1–6.

[8] Z. F. Siddiqui, E. Tiakas, P. Symeonidis, M. Spiliopoulou, and Y. Manolopoulos, "xstreams: Recommending items to users with time-evolving preferences," in *Proc. 4th Int. Conf. Web Intell. Mining Semantics*, 2014, pp. 22:1–22:12.

[9] P. Matuszyk, J. Vinagre, M. Spiliopoulou, A. M. Jorge, and J. Gama, "Forgetting techniques for stream-based matrix factorization in recommender systems," *Knowl. Inf. Syst.*, vol. 55, pp. 275–304, 2018.

[10] P. Pu, L. Chen, and R. Hu, "Evaluating recommender systems from the user's perspective: Survey of the state of the art," *User Modeling User-Adapted Interact.*, vol. 22, no. 4–5, pp. 317–355, 2012.

[11] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne, "Controlled experiments on the web: survey and practical guide," *Data Min. Knowl. Discovery*, vol. 18, no. 1, pp. 140–181, 2009.

[12] G. Hulten, L. Spencer, and P. M. Domingos, "Mining time-changing data streams," in *Proc. 7th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2001, pp. 97–106.

[13] J. Gama, R. Sebastião, and P. P. Rodrigues, "On evaluating stream learning algorithms," *Mach. Learn.*, vol. 90, no. 3, pp. 317–346, 2013.

[14] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *Proc. 8th Int. Symp. Intell. Data Anal.*, 2009, pp. 249–260.

[15] E. Diaz-Aviles, L. Drumond, L. Schmidt-Thieme, and W. Nejdl, "Real-time top-n recommendation in social streams," in *Proc. 6th ACM Conf. Recommender Syst.*, 2012, pp. 59–66.

[16] M. Blondel, Y. Kubo, and N. Ueda, "Online passive-aggressive algorithms for non-negative matrix factorization and completion," in *Proc. 17th Int. Conf. Artif. Intell. Statist.*, 2014, pp. 96–104.

[17] Y. Huang, B. Cui, J. Jiang, K. Hong, W. Zhang, and Y. Xie, "Real-time video recommendation exploration," in *Proc. Int. Conf. Manage. Data*, 2016, pp. 35–46.

[18] T. Yu, O. J. Mengshoel, A. Jude, E. Feller, J. Forgeat, and N. Radia, "Incremental learning for matrix factorization in recommender systems," in *Proc. Int. Conf. Big Data*, 2016, pp. 1056–1063.

[19] X. He, H. Zhang, M. Kan, and T. Chua, "Fast matrix factorization for online recommendation with implicit feedback," in *Proc. 39th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2016, pp. 549–558.

[20] J. Yin *et al.*, "Online collaborative filtering with implicit feedback," in *Proc. 24th Int. Conf. Database Syst. Advanced Appl.*, 2019, pp. 433–448.

[21] P. Liu, L. Zhang, and J. A. Gulla, "Real-time social recommendation based on graph embedding and temporal context," *Int. J. Hum.- Comput. Stud.*, vol. 121, pp. 58–72, 2019.

[22] K. Subbian, C. C. Aggarwal, and K. Hegde, "Recommendations for streaming data," in *Proc. 25th ACM Int. Conf. Inf. Knowl. Manage.*, 2016, pp. 2185–2190.

[23] X. Huang, L. Wu, E. Chen, H. Zhu, Q. Liu, and Y. Wang, "Incremental matrix factorization: A linear feature transformation perspective," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, 2017, pp. 1901–1908.

[24] J. Vinagre, A. M. Jorge, and J. Gama, "Fast incremental matrix factorization for recommendation with positive-only feedback," in *Proc. 22nd Int. Conf. User Modeling Adaptation Personalization*, 2014, pp. 459–470.

[25] R. Pálovics, A. A. Benczúr, L. Kocsis, T. Kiss, and E. Frigó, "Exploiting temporal influence in online recommendation," in *8th ACM Conf. Recommender Syst.*, 2014, pp. 273–280.

[26] A. M. Yagci, T. Aytekin, and F. S. Gürgen, "Scalable and adaptive collaborative filtering by mining frequent item co-occurrences in a user feedback stream," *Eng. Appl. AI*, vol. 58, pp. 171–184, 2017.

[27] C. Lin, L. Wang, and K. Tsai, "Hybrid real-time matrix factorization for implicit feedback recommendation systems," *IEEE Access*, vol. 6, pp. 21 369–21 380, 2018.

[28] P. Matuszyk, J. Vinagre, M. Spiliopoulou, A. M. Jorge, and J. Gama, "Forgetting methods for incremental matrix factorization in recommender systems," in *Proc. 30th Annu. ACM Symp. Appl. Comput.*, 2015, pp. 947–953.

[29] T. Kitazawa, "Sketching dynamic user-item interactions for online item recommendation," in *Proc. Conf. Conf. Hum. Inf. Interact. Retrieval*, 2017, pp. 357–360.

[30] M. Al-Ghossein, T. Abdessalem, and A. Barré, "Dynamic local models for online recommendation," in *Proc. Companion Web Conf.*, 2018, pp. 1419–1423.
[31] M. Al-Ghossein, P. Murena, T. Abdessalem, A. Barré, and A. Cornuéjols, "Adaptive collaborative topic modeling for online recommendation," in *Proc. 12th ACM Conf. Recommender Syst.*, 2018, pp. 338–346.
[32] J. Vinagre, A. M. Jorge, and J. Gama, "An overview on the exploitation of time in collaborative filtering," *Wiley Interdisciplinary Reviews, Data Mining Knowl. Discovery*, vol. 5, no. 5, pp. 195–215, 2015.
[33] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 6, pp. 734–749, Jun. 2005.
[34] P. G. Campos, F. Díez, and I. Cantador, "Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols," *User Model. User-Adapted Interact.*, vol. 24, no. 1–2, pp. 67–119, 2014.

João Vinagre received the PhD degree from the joint doctoral programme by the Universities of Minho, Aveiro, and Porto. He is currently a researcher at the Laboratory of Artificial Intelligence and Decision Support (LIAAD) of INESC TEC and an invited professor at the Faculty of Sciences of the University of Porto (FCUP). His research focuses on recommender systems and user modeling, with special emphasis on stream-based algorithms, evaluation issues, and incremental ensemble models. He has published several journal and conference papers on these subjects, and co-organized two editions of ORSUM, the workshop on Online Recommender Systems and User Modeling, held at The Web Conference 2018 and ACM RecSys 2019.

Alípio Mário Jorge received the PhD degree in computer science from U. Porto. He is currently an associate professor with the Department of Computer Science of the Faculty of Science of the U. Porto and the coordinator of LIAAD/INESC TEC, the Artificial Intelligence and Decision Support Lab of U. Porto since 2012. His research interests include the data mining and machine learning, in particular recommender systems, NLP, and web intelligence. He lectures on information processing and data mining. He leads research projects on data mining and Web intelligence. He has co-chaired international conferences (ECML/PKDD 2005, Discovery Science 2009, ECML/PKDD 2015), workshops and seminars in data mining and artificial intelligence. For more information, please visit http://www.dcc.fc.up.pt/∼amjorge.

Conceição Rocha received the PhD degree in applied mathematics from the University of Porto, Portugal. She is currently a researcher at INESC TEC. Her main interests include data validation, data collecting techniques, modelling, statistics, predictive modelling, classication, and text mining/data mining. She has published some journal and conference papers on diverse subjects, and co-organized three workshops: the last three national workshops on Classification and Data Analysis (JOCLAD 2017, 2018 and 2019) and 2018 international workshop in Symbolic Data Analysis (SDA2018).

João Gama received the PhD degree in computer science from the University of Porto, Portugal. He is currently an associate professor at the University of Porto and a senior researcher at LIAAD Inesc Tec. His main interests include Machine Learning, data mining, mainly in the context of data streams. He published more than 200 papers in major International conferences and journals, served as chair at ECML05, DS09, ADMA09, IDA11, and ECML PKDD 2015. He co-organized a series of workshops on learning from data streams in conjunction with ECML PKDD, KDD, SAC, and ICML. He is a member of the editorial board of MLJ, DAMI, NGC, KAIS, and PAI, and has authored a recent book in knowledge discovery from Data Streams.