Are We Forgetting Something? Correctly Evaluate a Recommender System With an Optimal Training Window

Robin Verachtert[1,2], Lien Michiels[1,2] and Bart Goethals[1,2,3]

[1]*Froomle N.V., Belgium*

[2]*University of Antwerp, Antwerp, Belgium*

[3]*Monash University, Melbourne, Australia*

Abstract

Recommender systems are deployed in dynamic environments with constantly changing interests and availability of items, articles and products. The hyperparameter optimisation of such systems usually happens on a static dataset, extracted from a live system Although it is well known that the quality of a computed model highly depends on the quality of the data it was trained on, this is largely neglected in these optimisations. For example, when concept drift occurs in the data, the model is likely to learn patterns that are not aligned with the target prediction data. Interestingly, most scientific articles on recommender systems typically perform their evaluation on entire datasets, without considering their intrinsic quality or that of their parts. First, we show that using only the more recent parts of a dataset can drastically improve the performance of a recommendation system, and we pose that it should be a standard hyperparameter to be tuned prior to evaluation and deployment. Second, we find that comparing the performance of well-known baseline algorithms before and after optimising the training data window significantly changes the performance ranking.

1. Introduction

Recommendation systems are widely used to help users find the most relevant products and articles from the large catalogues available on most websites, like news websites and e-commerce shops. The environments in which they are deployed generate large volume information streams on which the models need to be trained. Barring online learning methods and incremental models, the usual approach is to take a static slice of this data stream and train the model on this slice. Determining the optimal width of this slice is a challenging engineering problem. Using too little data can cause the model to starve, and not learn anything relevant. Using more data often results in longer training times, and larger models that take longer to predict.

In academic research, however, this is usually not considered to be such an issue. The typical datasets used for experimental evaluation are static, and they are almost always used in their entirety. Important steps have been taken to correctly evaluate recommendation techniques

CEUR Workshop Proceedings (CEUR-WS.org)

through temporal or leave-last-one-out splits [1, 2, 3, 4]. By using all historic events to train models, however, these evaluations place an implicit trust on the earliest interactions in the dataset to add useful information. Challenging this trust, algorithms have been designed to diminish the impact of older interactions [5]. In the evaluation of algorithms, we show that disregarding earlier interactions entirely during training can significantly improve the performance of a recommender system in multiple settings. Intuitively, this is true for a simple popularity baseline: Items that were popular in the past week are more predictive of next week than items that were popular in the past year [6]. But is this also true for more complex, personalised recommendation algorithms?

In this paper, we consider the maximum 'age' of an interaction, i.e. the time since it occurred, used to build the model an additional hyperparameter during model training. In the remainder of this paper, we will refer to the maximal age of an event used in training as the hyperparameter δ.

We investigate and answer the following three questions:

- *RQ1: How does the optimisation of δ impact the individual performance of an algorithm?*
- *RQ2: Does the optimisation of δ change the relative performance between the algorithms?*
- *RQ3: How does the choice of δ impact secondary metrics such as run time and coverage of the item catalogue?*

Additionally, through our experiments we show that the optimal δ has a significant impact on model accuracy across algorithms and datasets. The biggest improvements are found for algorithms that are agnostic of time, especially when deployed in highly dynamic environments such as online news. Our findings strengthen our conviction that the hyperparameter δ is an important consideration when determining which model performs best, both in future academic research and production settings. We leave a comprehensive benchmark of algorithms with optimal values of hyperparameter δ to future work.

In Section 2 we highlight relevant related work. Section 3 describes how δ should be considered a hyperparameter, and how to set up an evaluation to mimic a real-world scenario. Also in Section 3 we present the chosen algorithms, datasets and evaluation metrics. Finally, Section 4 discusses the experimental results with regards to the three research questions and presents the results from two trials on news websites confirming our results. We also use our experiments to give suggestions for the selection of values for δ.

2. Related Work

Research in data science has recognised that data drift is an import factor in training high quality models for several decades [7, 8, 9, 10] More specifically, Fan [11] raised awareness for the issues associated with the blind usage of older data in the context of binary classification. As they conclude: "[...] using old data unselectively is like gambling". When a dataset contains drift and an algorithm is not equipped to deal with this drift, using only more recent data, i.e. explicitly defining δ, is a straightforward way to avoid training poorly performing models [11].

Recommender systems are used in highly dynamic environments and so naturally have to deal with data drift. We can distinguish between two research directions related to handling data drift, i.e. measuring accuracy under data drift and recommendation algorithms that perform

well under data drift. Regarding the former, improved data splitting techniques that better reflect realistic recommendation scenarios have been proposed, e.g. timed splits [12, 13], a sequential last item prediction split [14] and repeated time-aware splitting [15, 1]. In relation to the latter, a large amount of time- and sequence-aware algorithms have been proposed over the years. For a comprehensive overview we refer the interested reader to Campos et al. [12], Ludewig and Jannach [16], Quadrana et al. [17] and Bogina et al. [5].

Relevant to our work, Vinagre and Jorge [18] summarised two generic methods for dealing with concept drift in a data-stream. The first is to utilise a predetermined δ and use it as a sliding window over the data. The second is to utilise fading factors such that older interactions have less influence on the similarities. Ludmann [19] used a contextual popularity algorithm, with δ equal to five minutes, thirty minutes and one hour, to great success in the CLEF Initiative in 2017. Similarly Ji et al. [6] showed that computing popularity using a small δ or using fading factors provided a much stronger baseline. Jannach and Ludewig [20] and Jannach et al. [14] find similar indications that the recency of training data is important in a retail context. Our work is inspired by these earlier efforts and aims to further anchor and broaden their findings regarding popularity and similarity-based algorithms to other types of recommendation algorithms, such as time- and session-aware algorithms. Examples of such time-aware algorithms are neighbourhood-based models that use fading factors [21, 22, 23, 24, 25, 26], similar to Vinagre and Jorge [18]. More recently, we see sequence- and session-aware algorithms that learn sequential models utilising the order in user histories. Examples of such methods are STAN [27], Sequential Rules [20], VS-KNN [20], and GRU4Rec [28]. In the wake of GRU4Rec, more and more deep learning approaches have been proposed that incorporate sequential and/or temporal information. [e.g. 29, 30, 2].

Recent reproducibility studies have challenged the performance of these complex deep learning methods in a variety of domains. In two recent works, Dacrema et al. [31, 32] found that "11 of the 12 reproducible neural approaches can be outperformed by conceptually simple methods", such as ItemKNN or UserKNN. Ludewig et al. [33] investigated the performance of deep learning approaches compared to simpler baselines in a session context. They found that "In the majority of the cases [...] it turned out that simple techniques outperform recent neural approaches". We follow their results, and focus on simpler baselines in our experiments.

3. Methodology

3.1. Recommendation Scenario

In many real-world applications, recommendation systems are used to generate recommendations for users while they are looking at other articles or products. In these use cases, the interest of the user is often captured mostly by their most recent interactions. A standard evaluation protocol to model this situation is to perform either leave-last-one-out splits [2, 3, 4], or iterative revealing [33].

We modify the leave-last-one-out evaluation to best approximate a (repeated) train-and-serve architecture typically used in production settings and avoid leaking future information into our model training [34]. Only data before a timestamp T, the time at which the model is supposedly retrained before serving, is used for training. Given the significant computational cost to

running our experiments, we use a single evaluation window and leave repeated evaluation as suggested by Scheidt and Beel [15] for future work.

Formally, given a set of users U and a set of items I, let $\mathcal{D} = \{(u, i, t) : u \in U, i \in I, t \in \mathbb{N}\}$ be the dataset of interactions, where t is the timestamp on which user u last interacted with item i.

To obtain a training dataset, we split the dataset on timestamp T; data before T ($\mathcal{D}_{<T}$) is used as training data for the algorithms. In addition to the other hyperparameters of each algorithm, we introduced the hyperparameter δ and so further limit our dataset used in training to data after $T - \delta$, i.e. $\{(u, i, t) \in \mathcal{D}_{<T} | T - \delta < t < T\}$. Small values for δ limit the training data to only interactions that occurred close to the cut-off timestamp T. The larger δ becomes, the more data is used to train the models.

To create the test dataset we extract only users that have at least 1 event after T and use all but their last interaction (also those before T) as history to predict their last interaction, as in a classical leave-last-one-out scenario.

To properly tune hyperparameters we introduce a second cut-off timestamp $T_{val} < T$, such that our training dataset during hyperparameter optimisation are the events $\{(u, i, t) \in \mathcal{D}_{<T_{val}} | T_{val} - \delta < t < T_{val}\}$. To obtain the validation evaluation dataset, we extract users that have an interaction between $T_{val} < t < T$ and, analogous to the test dataset, use all prior interactions to predict these users' final interaction.

When predicting items for a user, both during validation and testing, we remove their previously visited items from the recommendations, as is frequently done in real world settings as well. We consider evaluating re-consumption in this setting as future work.

3.2. Datasets

For our experiments, we use five datasets, two from the news domain, and three from the retail domain. We chose these two domains, because they are stereotypical real-world recommendation use cases, and we expect the domains to exhibit different behavioural patterns. News has a pronounced concept drift as articles become irrelevant quickly, while in retail, product relevance is typically stable for a longer period. Intuitively we expect this to result in retail datasets benefiting from larger δ values as they experience weaker data drift, while performance on news datasets suffers more drastically when δ is too large. In our selection of datasets, we required them to be of sufficient size ($> 1M$ interactions) and to contain timestamp information for the item view events, which will be used to train models. For news, we use the Adressa dataset [35] as well as a proprietary dataset, extracted from a live recommender system, which we'll call NEWS. Both of these datasets were collected over 7 days. In splitting these datasets, we used the second to last day 12:00 to 23:59 as the source for the validation target dataset, and the last day from 12:00 to 23:59 for the test target dataset. For retail, we use the Yoochoose dataset from the Recsys Challenge in 2015 [36], the CosmeticsShop Kaggle dataset [37] and a second proprietary dataset, extracted from a live recommender system, which we'll call RETAIL. All three of these datasets span a longer period than the two news datasets, with CosmeticsShop collected over 152 days, Yoochoose over 182 days and RETAIL over 98 days. For the CosmeticsShop and Yoochoose datasets, we used validation and test sets of 14 days, for the slightly shorter but denser RETAIL dataset we used consecutive 7-day windows.

Table 1
Properties for the datasets used in the offline experiments

Dataset	$|\mathcal{D}|$	$|\mathbf{U}|$	$|\mathbf{I}|$	Period	Gini(item)
RETAIL	24 237 016	1 302 909	18 255	98d	0.70
Yoochoose	16 044 427	1 882 684	44 415	182d	0.76
CosmeticsShop	7 877 677	483 080	27 019	152d	0.60
NEWS	5 943 609	381 797	3 810	7d	0.87
Adressa	2 532 729	228 462	2 790	7d	0.92

By using proprietary datasets as well as public datasets, we can link the offline experimentation results to our online trials.

The properties of the datasets can be found in Table 1. We report the number of events ($|\mathcal{D}|$), number of users ($|U|$), number of items ($|I|$), the period during which data was collected, and the Gini coefficient comparing visits per item [38]. The Gini coefficient is a statistical measure of dispersion, and a high Gini coefficient indicates that a few items have the most interactions, and all the others are interacted with much less frequently. News datasets typically have a higher Gini coefficient, because every day only a few articles are relevant for all users.

3.3. Algorithms

We selected a combination of time-agnostic baseline algorithms, sequence-aware algorithms and a time-aware algorithm to compare the impact resulting from optimising δ for each of them.

Popularity   The most visited items are recommended to each user. Recommendations are only minimally personalised since items a user has interacted with before are removed from the recommendations as per the scenario (See Section 3.1).

Item-kNN   One of the most well-known and frequently used baseline algorithms for neighbourhood-based collaborative filtering [39, 40]. The model consists of a single matrix multiplication with an item-item matrix $\boldsymbol{S} \in \mathbb{R}^{|I| \times |I|}$: $\phi(\boldsymbol{X}) = \boldsymbol{X}\boldsymbol{S}$. Where, $\boldsymbol{S}_{i,j}$ holds the similarity between items i and j. The similarity metric to use is considered a hyperparameter. In our work we use cosine similarity and conditional probability as defined in Deshpande and Karypis [40]. Recent work on neural news recommendation highlights the remarkable competitiveness of simple neighbourhood-based methods compared to more complex alternatives [16, 41].

Item-kNN with fading factors   We use two versions of ItemKNN methods with fading factors. The first, proposed by Ding and Li [42], applies an exponential decay on user histories when using them for prediction. The item-item similarity matrix is computed exactly as it would be for the ItemKNN algorithm. The prediction function is changed to $\phi(\boldsymbol{X}) = w(\boldsymbol{X})\boldsymbol{S}$, where w applies an exponential decay on the interaction matrix $\boldsymbol{X}$. The decayed value for a user item interaction is $w_{u,i} = e^{-\alpha(t_0 - t_{u,i})}$, with t_0 representing now, and $t_{u,i}$ the last time user u visited item i and α is a hyperparameter. Despite the fading factor applied during

prediction, we consider this algorithm time-agnostic in our discussion, because the trained model is time-agnostic. We will call this method "IKNN Ding" in the remainder of the paper.

The second method presented by Liu et al. [22] applies an exponential decay function on the binary interaction matrix before computing the similarities S. The similarities S are computed as the cosine similarity between the columns of the decayed interaction matrix: $w(X)$. The decay function w is identical to the one used in IKNN, as is the prediction function ϕ. The hyperparameters α_{fit} and α_{predict} of the decay function can be chosen independently for training and prediction, allowing for more flexibility. We'll reference this algorithm as "IKNN Liu".

EASE$^{\text{r}}$ This model was proposed as an extension of the well-known SLIM method [43, 44]. In EASE$^{\text{r}}$, the item-item matrix S is found through a least-squares optimisation problem that allows for a closed-form solution. This makes the model much more efficient to compute than iteratively optimised alternatives like Neural Networks, whilst yielding highly competitive results. As the optimisation requires inverting the Gramian item-item matrix, EASE$^{\text{r}}$ becomes more costly to compute as the size of the item catalogue grows.

GRU4Rec The first deep learning model for recommendations to utilise a GRU component to model sequential patterns in a session or user's history [45]. The model was inspired by text analysis methods and aims to capture relations between words that frequently appear together in a particular order. In our experiments, we use the variant that with Bayesian Personalised Ranking (BPR) loss to optimise the model, rather than using cross-entropy loss. BPR is more suited for our scenario, because it solves a ranking problem and does not approach the problem as a binary classification task. In addition this loss is also more efficient to compute, so training times are lower.

Sequential Rules (SR) Baseline algorithm using sequential association rules between items. The model recommends items related to the last item a user has seen: $\phi(X) = X_l S$. X_l is the binary last visit matrix, $X_l(u, i) = 1$ only if i is the last item visited by user u. The asymmetric similarity between item i and j: $S_{i,j}$, is computed as $\sum_{u \in U} \frac{\mathbb{1}(u,i,j)}{\text{gap}(u,i,j)}$. Where $\mathbb{1}(u, i, j)$ is an indicator function, that returns 1 only if user u has seen item j after item i, and gap returns the number of steps required to go from i to j. A hyperparameter `max_steps`, specifies how big this gap can maximally be before the co-occurrence is ignored. Ludewig et al. [33] found that despite the simplicity of the algorithm, it performed competitively in sequential recommendation tasks.

3.4. Evaluation metric(s)

We consider the problem of optimal ranking of items, also known as the Top-K recommendation problem. We use Normalised Discounted Cumulative Gain (NDCG) [46], Catalog Coverage (Coverage) [47], Recall [46] and Mean Reciprocal Rank (MRR) [40] as metrics. The metrics were evaluated on the top K recommendations, with $K \in [10, 20, 50]$. The goal we set for our experiments is to generate an optimal ranking of items to be shown to the user as a list of

recommendations. Due to space considerations we report only NDCG@10 and Coverage@10 in this paper. Other results can be found in the public code repository[1].

Our primary metric is NDCG. We choose this metric because it rewards models that put the correct items higher in the list. Besides this primary metric, we also report the coverage of the algorithms because the amount of items recommended is often seen as a secondary goal for recommendation [48].

3.5. Parameter Optimisation

We determine the optimal hyperparameters for each algorithm and dataset combination by performing a search over the hyperparameter space and evaluating performance on the validation dataset.

Using a grid search, even one with coarse settings, would not be feasible given the large amount of parameters for some of the algorithms, and the further addition of δ to be inspected over a large range of potential values.

Rather than using a random search we utilised the Tree-structure Parzen Estimator [49] as implemented in the Python `hyperopt` library[2] [50]. While none of our hyperparameter spaces contains dependent hyperparameters, the approach still manages to find optimal hyperparameter combinations in fewer trials than a random search would.

We don't set a fixed amount of trials but give each algorithm-dataset pair a fixed amount of time to run trials in order to find the best parameters. All algorithms were given six hours to find the optimal hyperparameters, however, only GRU4Rec was unable to find convergence within this timeframe. All other methods converged much sooner, often in less than two hours. This way all experiments can be run in under a week without parallel computation on an 8-core virtual machine with 52 GB of RAM, and a single NVIDIA Tesla T4 GPU. Due to insufficient RAM, we could not train the EASE algorithm on Yoochoose and RETAIL datasets, and GRU4Rec on the RETAIL dataset.

In order to enable the exploration of more hyperparameters for GRU4Rec we did not train it to full convergence during optimisation. This might lead to a loss of performance in the optimisation results, however, the loss will be similar for every parameter combination so we can find the optimal parameter combination while saving time on each trial. For the final results on the test dataset, we train the GRU4Rec models for 20 epochs, resulting in convergence.

---

[1]https://github.com/verachtertr/short-intent
[2]https://hyperopt.github.io/hyperopt/

4. Results

dataset	RETAIL	Yoochoose	CosmeticsShop	NEWS	Adressa
EASE^r	-	-	389	3	3
GRU4Rec	-	733	1562	9	121
ItemKNN	877	228	2368	2	5
Popularity	3	25	286	1	1
SR	2059	185	2976	3	18
IKNN Ding	530	214	2278	2	5
IKNN Liu	2139	280	1939	3	117

In this Section we share the results of our experiments and answer the three research questions. To enable reproduction and reuse of our experiments, we have made the code repository public[3].

4.1. RQ1: "How does the optimisation of δ impact the individual performance of an algorithm?"

In Table 2 we present the optimal values for delta found during optimisation, and in Table 3 we present the corresponding NDCG@10 values. We compute an NDCG value for both the model trained on all training data ($\delta = \infty$) and on the optimised δ ($\delta =$ optim).

The optimal choice of δ depends on the combination of the dataset and the algorithm.

A popularity algorithm works best with only the most recent data. Its optimal training window is on most datasets smaller than a day, with only CosmeticsShop exhibiting stable enough behaviour for 10 days to be optimal. On the news datasets we find the most drastic improvements, up to 30 times on the Adressa dataset. The extraordinary performance of the Popularity algorithm on news datasets and Adressa in particular is explained by the extreme popularity bias present in these datasets. In Table 1 you find that for Adressa the Gini coefficient of the items is 0.92, and on the test dataset, the Gini coefficient is even higher: 0.98. This indicates that almost all events happen on a very small group of popular items.

On the news datasets, the relevance of recent data is reflected in the optimal δ values, the time agnostic methods perform optimally using the last few hours to train. Only the time-aware ItemKNN model (IKNN Liu) and GRU4Rec manage to use more than a day of data without losing quality on the Adressa dataset. For both datasets, we see noticeable improvements in performance for the time agnostic algorithms trained on only recent data. For the NEWS dataset, with even more rapidly changing relevance, we see that all algorithms, even the time-aware algorithm, perform optimally using only the last few hours of data.

On the retail datasets, we see their stability reflected in the optimal δ values. CosmeticsShop is a very stable dataset, and most algorithms perform optimally using almost all of the data (The maximal value for δ is $124 * 24 = 2976$ hours). For RETAIL, we note that the optimal δ is usually smaller than on CosmeticsShop, but the performance gains are minimal. This implies

---

[3]https://github.com/verachtertr/short-intent

Table 3

NDCG@10 in % for optimised δ values and $\delta = \infty$. At the bottom of the table we report the correlation between the ranking of algorithms trained with $\delta = \infty$ and the one with optimised δ.

| dataset | RETAIL | | Yoochoose | | CosmeticsShop | | NEWS | | Adressa | |
delta	∞	optim	∞	optim	∞	optim	∞	optim	∞	optim
EASE^r	-	-	-	-	4.84	4.60	2.01	5.47	0.82	6.98
GRU4Rec	-	-	13.57	13.61	3.30	2.93	3.67	3.15	4.06	3.87
ItemKNN	6.42	6.43	16.50	17.84	4.89	4.90	1.27	4.91	0.44	5.40
Popularity	0.71	0.82	0.36	1.12	0.88	1.07	0.95	4.82	0.37	12.57
SR	9.30	9.30	19.04	20.69	7.23	7.23	3.23	4.47	3.59	4.53
IKNN Ding	8.50	8.51	17.10	18.52	6.44	6.43	1.49	5.76	0.60	6.44
IKNN Liu	8.81	8.81	18.84	18.68	6.41	6.40	2.60	3.56	3.92	3.91
correlation	1.00		1.00		1.00		-0.43		-0.71	

that we can build a good model using less data, but adding the additional data does not hurt performance as much as it did in the news use case. Yoochoose is the retail dataset where optimisation of δ has the largest impact. Most algorithms perform best using somewhere around the last 10 days of data, only GRU4Rec requires a month of data to get the best model.

The GRU4Rec algorithm shows the most inconsistent behaviour between validation and testing data. The optimal values found during optimisation do not seem to translate to optimal performance during testing. One possible reason for this, is that the model takes much longer to train, and so far fewer parameter combinations could be checked.

Choosing δ right is important to get the optimal performance for an algorithm given a dataset. In some cases, the dataset will be stable enough that using all data is optimal. In others, however, it's only the last few hours that hold relevant events to build a model for the imminent future.

4.2. RQ2: "Does the optimisation of δ change the relative performance between the algorithms?"

We compare how the rankings of algorithms sorted by NDCG change if we go from $\delta = \infty$ to an optimised δ. For this comparison we use Kendall's Tau correlation between the two rankings of the algorithms [51]. We report these correlations at the bottom of Table 3. On the two news datasets, we note a strong dissonance between the rankings. Both have a correlation value below zero, indicating that the rankings have drastically changed. When $\delta = \infty$, the time- and sequence-aware approaches show superior performance, however, this is no longer the case given an optimal δ. The baseline methods surpass the deep learning methods and now perform the best.

For the retail datasets we don't see this effect, either $\delta = \infty$ is optimal (CosmeticsShop and RETAIL), or the time agnostic algorithms were already outperforming the deep learning methods, and their improvement only further established their rank (Yoochoose). There is however no guarantee that the rankings will always remain the same, we can imagine that for some combinations of algorithms this ranking would change. Especially when comparing time-aware models with time agnostic baselines. The time-aware models will have a higher

Table 4

Coverage@10 in % for optimised δ and using $\delta = \infty$. Reducing δ usually results in a lower coverage, as older items are no longer recommended.

| dataset | RETAIL | | Yoochoose | | CosmeticsShop | | NEWS | | Adressa | |
delta	∞	optim	∞	optim	∞	optim	∞	optim	∞	optim
EASE^r	-	-	-	-	60.86	56.81	34.12	24.78	23.19	13.91
GRU4Rec	-	-	71.52	52.75	70.02	66.84	41.00	18.53	34.52	32.69
ItemKNN	94.03	89.93	76.51	63.10	59.95	61.30	25.77	21.21	10.39	16.74
Popularity	0.22	0.17	0.07	0.13	0.20	0.15	3.70	1.50	1.94	0.90
SR	89.65	89.52	85.83	65.46	92.47	92.47	47.82	24.44	41.29	23.23
IKNN Ding	90.86	81.12	88.47	71.90	93.68	93.99	14.38	22.05	14.62	17.03
IKNN Liu	88.17	88.15	78.28	73.22	93.19	93.98	65.70	30.42	71.36	68.57

performance when using the whole datasets, and the baselines manage to close the gap when their training window is optimised. We can see this happen on Yoochoose, IKNN Ding's performance almost matches that of IKNN Liu with an optimised δ, when it was outperformed on the $\delta = \infty$ setting.

In most scientific articles, the results would be compared using a $\delta = \infty$ setting, and so the time agnostic algorithms can be handily beaten by methods that do manage to take into account the order and/or time of the interactions. However, simple baselines trained on the more relevant - recent - part of the data, become much harder to improve on and even perform best in some of our experiments. This highlights why it is so important to optimise δ. If we do not, we risk making the wrong conclusions.

4.3. RQ3: How does the choice of δ impact secondary metrics such as run time and coverage?

In Table 4 we present the Coverage@10 results for the algorithm-dataset pairs. We see that in general coverage is lower for the optimal δ. This is to be expected, because one of the side effects of using less data is that older articles have no events, and so will not get recommended. Only for ItemKNN and IKNN Ding on Adressa do we see an inverse effect: Shrinking the training window increased the number of items recommended. This behaviour occurs when the historic data drowns more recent interactions, such that even given the recent history of the user, the model still mostly recommends a select group of older items. Reducing δ levels the playing field for the more recent items, and so more of them can get recommended depending on the interests of the users.

A third metric impacted by the selection of δ is the run time of the algorithms. Training a model on less data usually leads to lower training and prediction times. We compute run time as the sum of training and prediction time, thus accounting for both slow training and slow prediction. Both are impacted by the amount of data used and both contribute to problematic situations in production settings. In Table 5, the run time for the optimisation trials with optimal δ and maximal δ are reported. Using less data leads to lower run times. For production settings, this is an important insight. For example, on the Yoochoose dataset using the SR algorithm,

Table 5

Runtimes (in seconds) for optimal and non optimised δ. Runtime is the sum of training and prediction time. Decreasing δ also decreases the runtime, as less data needs to be processed.

| dataset | RETAIL | | Yoochoose | | CosmeticsShop | | NEWS | | Adressa | |
delta	∞	optim	∞	optim	∞	optim	∞	optim	∞	optim
EASE^r	-	-	-	-	815	791	38	30	14	7
GRU4Rec	-	-	7233	2990	5649	3824	1850	451	809	699
ItemKNN	198	188	96	20	117	55	43	14	15	4
Popularity	33	28	32	27	12	10	17	15	6	6
SR	2504	538	953	94	959	722	572	26	158	26
IKNN Ding	174	126	105	52	116	82	33	16	14	4
IKNN Liu	194	67	100	57	128	87	44	16	19	11

there is a small increase in performance when changing to an optimal δ but also a 10-time reduction in run time. This means that models can be updated more frequently and with lower computational cost.

This highlights a final reason why using less data should be considered. When using as much data as is available, we not only risk lower performance, we are also incurring higher computational costs and creating larger delays when building models and generating recommendations.

4.4. Online Tests

Complementing the offline results, we also performed two online trials on different news websites. The goal for these trials was to optimise recommendation boxes that serve a list of popular items to the users. Before the use of an automated optimisation of δ, the training window was chosen manually by engineers with some input from editors. By performing the optimisation of δ as suggested in this work, we found that the original values were not optimal, and could be improved by using smaller δ values.

In a first test, using the website from which the NEWS dataset was extracted, the box was found on the homepage. The manual setting was to train every three hours. Thus $\delta = 3h$ was used as our control treatment. During the offline experiments, we found that $\delta = 1h$ performed optimally, and so for the test group we used this as the training window. The results of the AB test showed that the optimised $\delta = 1h$ training window resulted in an improvement in CTR (on the box) of 7% during a period of three days. After which we concluded the test, and enabled the new setting for all users. We could use a short testing window thanks to high traffic; Three million recommendation lists were generated for both groups combined. The 7% improvement we found online, is similar to the 10% improvement we found offline.

In a second test on a different news website, we found an optimal window of $\delta = 2h$ after parameter tuning. In this more extensive test, we deployed a similar recommendation list in multiple locations on the website to make sure the positive effects were consistent. Furthermore, the test was run for two weeks to allow for variations between days. We used two control groups, one with training window $\delta = 6h$, and a second with $\delta = 10h$. Depending on the location of the box we found an improvement in CTR of 7% to 8% over both control groups,

which performed nearly identical.

Even though these experiments were only done using a popularity-based algorithm, they show the value in optimising the δ parameter before deploying the algorithms in a production setting. The improvements we find in our offline experiments for this algorithm were reflected in our online experiments.

5. Conclusion

"Are we forgetting something?" we wrote in the title, and our answer is clearly: yes! When training and evaluating recommender systems, we typically forget to take the quality of the data into account, or even consider the use of only (the most) recent parts of the given datasets. As we have presented in this paper, the performance of state-of-the-art algorithms drastically changes when training only on a recent part of the data. Moreover, the performance ranking of state-of-the-art (both baseline and neural) algorithms changes significantly when using the optimal training window size δ. We believe that we have clearly shown that the choice of the δ matters, both to find the optimal performance of individual algorithms and to make a fair comparison between algorithms. Optimising δ for each algorithm should be standard practice in the evaluation of recommender systems. Not optimising δ will favour only the algorithms that account for drift.

6. Limitations and Future Work

In this work we focused on News and Retail datasets, as well as a selection of baseline algorithms. In future work we want to extend the experiments, by including additional relevant domains, such as entertainment, tourism and music, as well as using more recently presented state-of-the-art sequential recommendation methods. In doing so we want to provide a comprehensive benchmark of the state of the art in sequential recommendation.

Due to run time concerns we did not consider repeated evaluations. To solidify our findings, and make sure that they hold on more than one split, we aim to report results over time in future experiments.

This work's experiments focus on short term effects, extending these results to long term effects such as user retention is an interesting avenue for future research.

References

[1] O. Jeunen, K. Verstrepen, B. Goethals, Fair offline evaluation methodologies for implicit-feedback recommender systems with MNAR data, in: Proc. of the ACM RecSys Workshop on Offline Evaluation for Recommender Systems, REVEAL '18, 2018.

[2] C. Lonjarret, R. Auburtin, C. Robardet, M. Plantevit, Sequential recommendation with metric models based on frequent sequences, Data Mining and Knowledge Discovery 35 (2021) 1087–1133.

[3] I. Bayer, X. He, B. Kanagal, S. Rendle, A generic coordinate descent framework for learning from implicit feedback, in: Proc. of the 26th International Conference on World Wide Web, 2017, pp. 1341–1350.

[4] W.-C. Kang, J. McAuley, Self-attentive sequential recommendation, in: 2018 IEEE International Conference on Data Mining (ICDM), IEEE, 2018, pp. 197–206.

[5] V. Bogina, T. Kuflik, D. Jannach, M. Bielikova, M. Kompan, C. Trattner, Considering temporal aspects in recommender systems: a survey, User Modeling and User-Adapted Interaction (2022) 1–39.

[6] Y. Ji, A. Sun, J. Zhang, C. Li, A re-visit of the popularity baseline in recommender systems, in: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2020, pp. 1749–1752.

[7] A. Bifet, R. Gavalda, Learning from time-changing data with adaptive windowing, in: Proc. of the 2007 SIAM international conference on data mining, SIAM, 2007, pp. 443–448.

[8] R. Klinkenberg, I. Renz, Adaptive information filtering: Learning in the presence of concept drifts, Learning for Text Categorization (1998) 33–40.

[9] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Computing Surveys 46 (2014).

[10] G. Widmer, M. Kubat, Learning in the presence of concept drift and hidden contexts, Machine Learning 23 (1996) 69–101.

[11] W. Fan, Systematic data selection to mine concept-drifting data streams, in: Proc. of the 10th ACM SIGKDD international conference on Knowledge discovery and data mining, 2004, pp. 128–137.

[12] P. G. Campos, F. Díez, I. Cantador, Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols, User Modeling and User-Adapted Interaction 24 (2014) 67–119.

[13] U. Panniello, M. Gorgoglione, C. Palmisano, Comparing pre-filtering and post-filtering approach in a collaborative contextual recommender system: an application to e-commerce, in: International Conference on Electronic Commerce and Web Technologies, Springer, 2009, pp. 348–359.

[14] D. Jannach, M. Ludewig, L. Lerche, Session-based item recommendation in e-commerce: on short-term intents, reminders, trends and discounts, User Modeling and User-Adapted Interaction 27 (2017) 351–392.

[15] T. Scheidt, J. Beel, Time-dependent evaluation of recommender systems., in: Perspectives@ RecSys, 2021.

[16] M. Ludewig, D. Jannach, Evaluation of session-based recommendation algorithms, User Modeling and User-Adapted Interaction 28 (2018) 331–390.

[17] M. Quadrana, P. Cremonesi, D. Jannach, Sequence-aware recommender systems, ACM Computing Surveys (2018).

[18] J. Vinagre, A. Jorge, Forgetting mechanisms for scalable collaborative filtering, Journal of the Brazilian Computer Society 18 (2012) 271–282.

[19] C. A. Ludmann, Recommending news articles in the clef news recommendation evaluation lab with the data stream management system odysseus., in: CLEF (Working Notes), 2017.

[20] D. Jannach, M. Ludewig, When recurrent neural networks meet the neighborhood for session-based recommendation, in: Proc. of the 11th ACM Conference on Recommender

Systems, RecSys '17, ACM, 2017, pp. 306–310.

[21] T. Q. Lee, Y. Park, Y. Park, A time-based approach to effective recommender systems using implicit feedback, Expert systems with applications 34 (2008) 3055–3062.

[22] N. Liu, M. Zhao, E. Xiang, Q. Yang, Online evolutionary collaborative filtering, in: Proc. of the 4th ACM Conference on Recommender Systems, RecSys '10, ACM, 2010, pp. 95–102.

[23] C. Xia, X. Jiang, S. Liu, Z. Luo, Z. Yu, Dynamic item-based recommendation algorithm with time decay, in: 2010 Sixth International Conference on Natural Computation, volume 1, IEEE, 2010, pp. 242–247.

[24] Y. Liu, Z. Xu, B. Shi, B. Zhang, Time-based k-nearest neighbor collaborative filtering, in: 2012 IEEE 12th International Conference on Computer and Information Technology, IEEE, 2012, pp. 1061–1065.

[25] P. C. Vaz, R. Ribeiro, D. M. De Matos, Understanding the temporal dynamics of recommendations across different rating scales., in: UMAP Workshops, 2013.

[26] V. W. Anelli, T. Di Noia, E. Di Sciascio, A. Ragone, J. Trotta, Local popularity and time in top-n recommendation, in: European Conference on Information Retrieval, Springer, 2019, pp. 861–868.

[27] D. Garg, P. Gupta, P. Malhotra, L. Vig, G. Shroff, Sequence and time aware neighborhood for session-based recommendations: Stan, in: Proc. of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2019, pp. 1069–1072.

[28] B. Hidasi, A. Karatzoglou, L. Baltrunas, D. Tikk, Session-based recommendations with recurrent neural networks, arXiv preprint arXiv:1511.06939 (2015).

[29] J. Tang, K. Wang, Personalized top-n sequential recommendation via convolutional sequence embedding, in: Proc. of the 11th ACM international conference on web search and data mining, 2018, pp. 565–573.

[30] Q. Liu, Y. Zeng, R. Mokhosi, H. Zhang, Stamp: short-term attention/memory priority model for session-based recommendation, in: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, 2018, pp. 1831–1839.

[31] M. F. Dacrema, P. Cremonesi, D. Jannach, Are we really making much progress? a worrying analysis of recent neural recommendation approaches, in: Proc. of the 13th ACM Conference on Recommender Systems, RecSys '19, ACM, 2019, pp. 101–109.

[32] M. F. Dacrema, S. Boglio, P. Cremonesi, D. Jannach, A troubling analysis of reproducibility and progress in recommender systems research, ACM Transactions on Information Systems (TOIS) 39 (2021) 1–49.

[33] M. Ludewig, N. Mauro, S. Latifi, D. Jannach, Performance comparison of neural and non-neural approaches to session-based recommendation, in: Proc. of the 13th ACM conference on recommender systems, RecSys '19, 2019, pp. 462–466.

[34] Y. Ji, A. Sun, J. Zhang, C. Li, A critical study on data leakage in recommender system offline evaluation, arXiv preprint arXiv:2010.11060 (2020).

[35] J. A. Gulla, L. Zhang, P. Liu, O. Özgöbek, X. Su, The adressa dataset for news recommendation, in: Proc. of the International Conference on Web Intelligence, WI '17, ACM, 2017, p. 1042–1048.

[36] D. Ben-Shimon, A. Tsikinovsky, M. Friedmann, B. Shapira, L. Rokach, J. Hoerle, Recsys challenge 2015 and the yoochoose dataset, in: Proc. of the 9th ACM Conference on Recommender Systems, RecSys '15, ACM, 2015, pp. 357–358.

[37] M. Kechinov, Cosmeticsshop e-commerce dataset, 2020. URL: https://www.kaggle.com/mkechinov/ecommerce-events-history-in-cosmetics-shop, accessed: 2022-07-26.

[38] J. Y. Chin, Y. Chen, G. Cong, The datasets dilemma: How much do we really know about recommendation datasets?, in: Proc. of the 15th ACM International Conference on Web Search and Data Mining, 2022, pp. 141–149.

[39] B. Sarwar, G. Karypis, J. Konstan, J. Riedl, Item-based collaborative filtering recommendation algorithms, in: Proc. of the 10th International Conference on World Wide Web, WWW '01, ACM, 2001, pp. 285–295.

[40] M. Deshpande, G. Karypis, Item-based top-n recommendation algorithms, ACM Transactions on Information Systems (TOIS) 22 (2004) 143–177.

[41] G. D. S. P. Moreira, D. Jannach, A. M. da Cunha, Contextual hybrid session-based news recommendation with recurrent neural networks, IEEE Access 7 (2019) 169185–169203.

[42] Y. Ding, X. Li, Time weight collaborative filtering, in: Proc. of the 14th ACM international conference on Information and knowledge management, 2005, pp. 485–492.

[43] X. Ning, G. Karypis, Slim: Sparse linear methods for top-n recommender systems, in: Proc. of the 2011 IEEE 11th International Conference on Data Mining, ICDM '11, IEEE Computer Society, 2011, pp. 497–506.

[44] H. Steck, Embarrassingly shallow autoencoders for sparse data, in: Proc. of the World Wide Web Conference, WWW '19, ACM, 2019, p. 3251–3257.

[45] C. Wu, A. Ahmed, A. Beutel, A. J. Smola, H. Jing, Recurrent recommender networks, in: Proc. of the 10th ACM international conference on web search and data mining, 2017, pp. 495–503.

[46] G. Shani, A. Gunawardana, Evaluating recommendation systems, in: Recommender systems handbook, Springer, 2011, pp. 257–297.

[47] S. A. Puthiya P., N. Usunier, Y. Grandvalet, A coverage-based approach to recommendation diversity on similarity graph, in: Proceedings of the 10th ACM Conference on Recommender Systems, 2016, pp. 15–22.

[48] M. Ge, C. Delgado-Battenfeld, D. Jannach, Beyond accuracy: evaluating recommender systems by coverage and serendipity, in: Proc. of the fourth ACM conference on Recommender systems, RecSys '10, 2010, pp. 257–260.

[49] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, Advances in neural information processing systems 24 (2011).

[50] J. Bergstra, D. Yamins, D. Cox, Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, in: International conference on machine learning, volume 28, PMLR, 2013, pp. 115–123.

[51] M. G. Kendall, The treatment of ties in ranking problems, Biometrika 33 (1945) 239–251.