Temporal Dynamics in Online Recommender Systems

Robin Verachtert

Supervisor **prof. dr. B. Goethals**

Thesis submitted in fulfilment of the requirements for the degree of doctor of science: computer science
Faculty of Science | Antwerpen, 24/04/2023

University of Antwerp

Faculty of Science

Temporal Dynamics in Online Recommender Systems

Thesis submitted in fulfilment of the requirements for the degree of
doctor of science: computer science
at the University of Antwerp

Robin Verachtert

Antwerpen, 24/04/2023

Supervisor
prof. dr. B. Goethals

Jury

Chairman
prof. dr. F. Geerts, University of Antwerp, Belgium

Supervisor
prof. dr. B. Goethals, University of Antwerp, Belgium

Members
prof. dr. K. Laukens, University of Antwerp, Belgium
prof. dr. T. Calders, University of Antwerp, Belgium
dr. K. Verstrepen, Froomle, Belgium
prof. dr. K. Verbert, KU Leuven, Belgium
prof. dr. C. Vens, KU Leuven, Belgium

Contact
Robin Verachtert
University of Antwerp
Faculty of Science
Adrem
Groenenborgerlaan 171, 2020 Antwerpen, België
M: robin.verachtert@uantwerpen.be

Dutch title:

Tijdsaspecten in Online Recommender Systemen

Acknowledgements

This thesis is the result of four years of work. Four years, during which I have had the fortune to have been helped and supported by fantastic people. Without their help, motivation and friendship, I would not have been able to finish this work.

A crucial factor in keeping motivation and creativity has been to have good recreational activities. To all my friends at KSC OLVE, in particular, my teammates from "De Vier", and to Los Sportos, thank you for creating many fun afternoons and evenings, the perfect moments to relax and recharge before work. To everyone that went walking, climbing and running with me in the past years, thanks for keeping me active and healthy.

Doing my research supported by a Baekeland grant, I had the pleasure of working with colleagues from both Froomle and the Adrem Research Group. To my colleagues from the ADREM research group, thank you for discussing research topics, papers, results, accepted papers, rejected papers and many more. I am grateful to Froomle for the opportunity, I would most likely not have pursued a doctorate, were it not for the unique circumstances of being able to do it in such a practical environment. Thanks to everyone who has helped me build research capabilities into our product, and supported me with experiments. Thanks as well for all of the random donut calls, coffee moments and lunch discussions. Good colleagues make life so much nicer, and a simple chat makes the day better. Thanks to Koen and Bart for presenting the opportunity to do a PhD at Froomle.

Papers are rarely written alone, and my work is no exception, none of it would have been possible without the help of my co-authors for the various papers. Lien, Jeroen, Olivier, Len, Andres, Kim and Bart, thank you for making the work better, making me a better writer and helping me find the proper knowledge to write my publications. Thank you, Bart, for your help and time as my promotor. While we may not always have agreed, our discussions usually lead to a deeper knowledge and better final results. I really appreciate your help in making this possible.

The best work is not done in isolation, and I was fortunate enough to have Lien as a team member on the research team at Froomle. Thanks, Lien for being a fantastic colleague and friend, for the endless discussions about research, the many papers we wrote together, and the countless experiments we designed together. Thank you for helping me become a better version of myself, and believing that I could succeed at the doctorate before I did myself.

Last, but certainly not least, thank you, Mom, Dad and Gil for putting up with me for the past four years, and helping me when I was stressed or annoyed. Often it was the simple things, like making delicious food, or a happy good morning, that helped me through a hard day. Your support has meant the world to me and kept me going when I was tired, or fed up with striving for a deadline.

Robin, Antwerpen, 2023

Contents

Publications

- Robin Verachtert, Lien Michiels and Bart Goethals. Are We Forgetting Something? Correctly Evaluate a Recommender System With an Optimal Training Window. In *Proceedings of the Perspectives on the Evaluation of Recommender Systems Workshop*, 2022.
- Lien Michiels, Robin Verachtert, Kim Falk and Bart Goethals. Abstract: Should Algorithm Evaluation Extend to Testing? We Think So... In *Proceedings of the Perspectives on the Evaluation of Recommender Systems Workshop*, 2022.[1]
- Lien Michiels, Robin Verachtert, and Bart Goethals. RecPack: An(other) Experimentation Toolkit for Top-N Recommendation using Implicit Feedback Data. In *Proceedings of the 16th ACM Conference on Recommender Systems (RecSys '22)*, 2022.
- Len Feremans, Robin Verachtert and Bart Goethals. A Neighbourhood-based Location- and Time-aware Recommender System. In *Workshop on Online Recommender Systems and User Modeling (ORSUM)*, 2022. [1]
- Robin Verachtert, Lien Michiels, Jeroen Craps and Bart Goethals. The Impact of a Popularity Punishing Hyperparameter on ItemKNN Recommendation Performance. In *Proceedings of the 45th European Conference on Information Retrieval (ECIR '23)*, 2023.
- Robin Verachtert, Olivier Jeunen and Bart Goethals. Scheduling on a Budget: Avoiding Stale Recommendations with Timely Updates. In *Machine Learning with Applications*, Volume 11, 2023.
- Lien Michiels, Robin Verachtert, Andres Ferraro, Kim Falk and Bart Goethals. A Framework and Toolkit for Testing the Correctness of Recommendation Algorithms. In *Special Issue on Perspectives on Recommender Systems Evaluation in ACM Transactions on Recommender Systems (TORS)*, 2023. [1]

[1]This paper is not used in this thesis.

Chapter 1

Introduction

Recommender systems have become a central component in our daily use of the Internet. Most websites host so much content, that without high-quality recommender and search engines, we would struggle to find the needle of relevant content in this haystack.

Tech giants like Facebook, Netflix, and Spotify, have popularized the concepts of personalised recommendations, by proposing new connections, "Recommended for you" lists or personalised playlists. We also find these giants as a driving force in academic research [11, 28, 88, 94, 100, 112, 117], as they have a strong incentive to improve their recommender systems. What distinguishes them further from most other companies that want to use recommender systems is the amount of resources and data they have available. For most companies operating at a smaller scale, many of the most advanced methods are too costly to train, and the margin they have over simpler methods is too small to use them.

Therefore, there is a need in recommender system research to also find methods that serve these small- or medium-sized companies [101, 120].

Inspired by the real-world experience encountered as a researcher at Froomle, this thesis addresses problems with a direct application to production settings. Further, we highly value the ability to serve high-quality recommendations to real users, without requiring high computational costs.

1.1 The Recommendation Problem

At its core, a recommender system is a function $\phi(x) \rightarrow R$, which takes a vector x and generates a set of relevant items R. x Can take many forms, depending on the application, for example, a binary history vector, a user feature vector, or a list of most recently visited items. Finding the right function ϕ is where most of the challenges lie with this problem. Usually, machine learning is used to learn the "best" function given stored data, although manual rule-based approaches still see use as well.

Typically recommendation approaches are categorised into 'Collaborative Filtering', 'Content Based' and 'Hybrid' approaches. 'Content Based' approaches use the metadata available about the items (topics, tags, images, ...) and metadata available about

users (age, country, ...) to construct the function ϕ. 'Collaborative Filtering' only uses the interactions or ratings of users, and tries to build prediction models that use the similarity patterns in this data. Examples include item- and user-based neighbourhood methods, matrix factorisation, and neural network approaches. 'Hybrid' approaches combine content-based and collaborative methods, to get improved recommendation performance.

In our work, we focus on the "Collaborative Filtering" problem. This choice comes from real-world experience at Froomle, where we find that curating metadata is a challenge, usually requiring additional engineering work as each company maintains its own metadata structure. User-item interactions, on the other hand, are much easier to standardise, which allows the same solution to be reused with minimal effort. Examples of interactions include a user visiting a page, the user clicking on a recommendation, and the user purchasing an item.

In collaborative filtering, there are usually two types of data that may be available. 'Explicit Feedback' is collected by requesting the users to rate an item, usually in the form of a rating given to an item. The challenge in getting this data is that the users need to explicitly take action to provide the ratings. Implicit feedback, on the other hand, can be logged while the users use the website as usual. This 'Implicit Feedback', is all the information we can collect about a user by analysing the actions they take while on a website, what pages they visit, what they buy, what they consider but do not buy, etc. This second data is usually more abundant, and less intrusive to the user to collect [96].

We will thus focus on 'Implicit Feedback' in our work, if we want to personalise the experiences for as many users as possible, 'Implicit Feedback' will give us more information about them than 'Explicit Feedback' can.

In the implicit feedback setting, the dataset usually consists of a list of triples: $\mathcal{D} = \{(u, i, t) | u \in U, i \in i, t \in \mathbb{N}^+\}$. Where U is the set of users, I the set of items, and t is a timestamp.

For more efficient computation this dataset can also be represented in various ways depending on the type of algorithm.

The most common representation is to represent the data as a binary matrix $\boldsymbol{X}$, where each row indicates a user and each column indicates an item. When a user u has interacted with an item i, then $\boldsymbol{X}_{u,i} = 1$.

In works that utilise the timestamp of when a user interacted with an item, we construct a similar matrix $\boldsymbol{T}$ with the last timestamp of interaction for each (u, i) pair.

Other methods rely on sequential information and/or model repeated consumption. For these, the dataset is sorted per user, such that we have a list of interactions per user sorted by timestamp, ascending

1.2 Recommendations in Production

When going from theory to practice, the recommender system increases in complexity. Theoretical research usually focuses on algorithms, and metrics, the two components to finding ϕ. In production settings, these two are smaller components in a large recommender system, that also needs to handle data collection, real-time data fetching, timely serving of recommendations and more. So when we consider recommendations in a production setting, we don't just need to find a model ϕ that optimises a metric, we also need good operational quality of the model. Of particular interest to this work are:

- Training time: in many cases, we need to keep the models up to date by retraining them. When models take too long to train, they can be outdated by the time they are trained, or the cost for training them can be too high to be profitable
- Response time: users do not want to wait seconds before they see their recommendations, this means that our recommender system needs a way to respond, typically below 50ms [71].
- Model size: Large models require large amounts of resources to serve recommendations, and so can become prohibitively expensive.

These constraints inspire our choice to focus on simple algorithms relying on Nearest Neighbours or efficient exact solutions, rather than deep learning techniques.

Production settings also differ from the traditional recommendation problem presented above, in that the data is much more dynamic than the static representation $\mathcal{D}$ used in most research. New data is collected constantly as users interact with the website and thus also the recommendations. This need to handle change in the data inspired Chapters 3, 4 and 5, all dealing with dynamic datasets and presenting different ways in which to allow recommendation algorithms to exploit these dynamics.

Finally, production settings also imply collaboration with business stakeholders, who might want to see specific behaviour, such that the algorithms need to be adapted to meet their business needs. One of the important business needs that we investigate in this work is to activate as much of the available catalog as possible. Recommendation algorithms have a tendency to recommend mostly popular articles [2], this can lead to starvation of the other items, and fairness concerns in many use cases. We investigate this thoroughly in Chapter 6 and keep it in mind when evaluating algorithms in all of our experiments.

1.3 Evaluation

Recommender systems are a complex and challenging area of research, because of the different problems we use them for. Recommending items a user might be interested in after reading an article, is a very different problem from helping them discover new interests, which is also very different from looking for recommendations that maximise revenue on a retail website. Each separate problem will most likely require a different

function ϕ to perform optimally. So finding the right ϕ is not just a "do it once" problem, it is a fundamental problem in recommender systems research as well as application. Finding ϕ is usually done through the evaluation of many candidate functions, some that have proven value before in the same or similar fields, and others that are newly developed.

Correct evaluation is therefore an important factor in finding the right model to use for the problem at hand. Evaluation usually happens in three ways, online tests, user studies and offline tests [139]. In our work, we only focus on offline and online tests, due to the methods and data at our disposal.

1.3.1 Online Evaluation

Arguably the best way to evaluate the performance of an algorithm is to perform a controlled trial with real users that interact with the recommendations.

The most common approach to this is to use an A/B-test, which exposes a small part of the users to the algorithm or system to be tested, and the large remainder of users are served by a control system that was in place before, or which consists of a simple solution if no solution was in place before. Only once this AB test is successful, does the new algorithm or system get pushed to all users.

To evaluate performance online, different metrics are available, depending on the goal of the business deploying the recommender system. A commonly used metric is Click-Through-Rate (CTR), which measures the percentage of recommendation opportunities that also lead to a click. This metric has been criticised for potentially leading to short-term optimisation, rather than long-term impact, and other metrics such as session length, number of returning users or number of subscription conversions have been suggested to better measure business impact. In this thesis, we evaluate our A/B-tests using CTR. We choose this metric as it can be easily measured for a single list of recommendations, meaning we don't need to take into account other confounding factors (such as different tests, or interface changes elsewhere during the experiment). The only difference between the control and treatment groups is which items are recommended.

Online evaluations give us an insight into the real performance of the algorithm, but they also have several important limitations and costs.

- Online tests are expensive. Before an algorithm can be exposed to real users, it needs to be implemented in such a way that it meets all non-functional requirements. These include low response times, observability should an error occur, stability, robustness to bad data, and more. The engineering cost to get an algorithm to this point is prohibitively expensive to do for every idea we would like to test.
- The new algorithm or system can negatively impact business metrics and values when it performs poorly. Worse, this can extend beyond the test duration when users leave the system due to a bad experience.
- Online tests are also very narrow. Usually, the new test is performed in just a few spots on a website, and so the context for which we have results is also narrow.

This prevents us from making generalizations about performance in other use cases, without running multiple online trials.

- Online experiments are only available to the few researchers that have access to a platform that serves recommendations to an audience.

These limitations naturally create an opportunity and necessity for offline experiments, as a vetting process. Only the most promising approaches, that fulfil all functional and non-functional requirements should be tested online.

1.3.2 Offline Evaluation

In offline evaluation, the experiments are based on a fixed dataset, usually a snapshot of a website's traffic. Multiple algorithms are evaluated on a selection of metrics, and traditionally the metrics are then compared to find if a new algorithm outperforms a selection of state-of-the-art algorithms.

Different types of offline evaluation exist, depending on the type of algorithm evaluated, and the type of data available.

- *Off-policy evaluation* aims to estimate the performance of hypothetical policies using data generated by a deployed policy [111].
- *Simulators* are abstractions of online tests, in which users are modelled as a stochastic click process [9, 109]. These allow the simulation of interactive environments, without needing to expose the users to the recommendations. The approaches are limited by the quality and accuracy of the simulator.
- *Train-Test-Validation split based evaluation*, follows approaches known in classification and information retrieval, where the dataset is split into training, validation and test datasets, and models are evaluated on how well they find the items in the test (or validation set) [47, 139].

In this work, we focus on the train-test-validation split evaluation, and in Chapter 2, we present an open-source framework for experimentation `RecPack`, which we have built to support correct and efficient offline evaluation both in industry, and academia. In this chapter, we also provide an overview of the offline evaluation methodology used throughout this thesis, and some suggestions for best practices in experimental design.

1.4 Contributions

- In Chapter 2 we present the recommendation package `RecPack`. This package contains all the building blocks to make experiments reproducible as well as several implemented algorithms and base classes to enable fast experimentation for researchers. This package was presented in a demo paper at the Recsys '22: Lien

Michiels, Robin Verachtert, and Bart Goethals. RecPack: An(other) Experimentation Toolkit for Top-N Recommendation using Implicit Feedback Data. In *Proceedings of the 16th ACM Conference on Recommender Systems (RecSys '22)*, 2022. In addition to presenting the package, we also give some suggestions for how to evaluate recommendations based on our experiments.

- In Chapter 3 we present and compare four different methods to schedule a model update. We show that model updates are an important measure to counteract concept drift and maintain high-quality recommendations. We find that scheduling techniques that look at information gathered since the last update perform better than traditional time-based schedules. Through several online trials, we also confirm that we can use them in online scenarios to improve the recommendations by keeping models better up to date. This work was published in the Journal "Machine Learning with Applications": Robin Verachtert, Olivier Jeunen and Bart Goethals. Scheduling on a Budget: Avoiding Stale Recommendations with Timely Updates. In *Machine Learning with Applications*, Volume 11, 2023.

- In Chapter 4 we investigate how we can improve baseline models, by training them on less data. A common assumption is that more data will lead to better models. However, we notice that for baseline models, it is often important to train them only on the relevant part of the training data, which is strongly related to the target data. In traditional evaluations, popularity is computed on the whole dataset, and then used as a baseline. Popularity changes through time though, and so computing it only on the interactions of the last hour before the prediction period, makes it a much harder to beat baseline. Similar patterns can be observed for ItemKNN and other common baseline methods. This work was presented and published in the "Perspectives on Recommendation Workshop": Robin Verachtert, Lien Michiels and Bart Goethals. Are We Forgetting Something? Correctly Evaluate a Recommender System With an Optimal Training Window. In *Proceedings of the Perspectives on the Evaluation of Recommender Systems Workshop*, 2022.

- Chapter 5 continues the work started in Chapter 4, and looks at how the older data can still be useful to the models, thus improving on the naive data removal method. We provide an overview of the existing work on Time-Aware ItemKNN models and provide a framework that presents new combinations of options hitherto not presented in the literature. These new combinations manage to outperform most baselines that were trained on optimised training windows. This shows that the old data is not irrelevant, but needs to be used correctly. This work is under submission at the "Transactions on Recommender Systems" journal.

- In Chapter 6 we investigate a phenomenon we noticed in several of our experiments for previous chapters. In many of these we noticed that algorithms favouring popular items, such as popularity, or ItemKNN using conditional probability were scoring higher than expected on offline metrics, and these results were not reflected in online experiments. In a series of experiments we investigated how punishing popular items in recommendations has an impact on both the fairness of exposure, as well as the recommendation quality. We found that there is a trade-off between the fairness of exposure and the quality of the recommendations and that the offline results did show more bias towards algorithms recommending more popular items compared to the real user interactions. This work was presented at ECIR '23: Robin Verachtert, Lien Michiels, Jeroen Craps and Bart Goethals. The

Impact of a Popularity Punishing Hyperparameter on ItemKNN Recommendation Performance. In *Proceedings of the 45th European Conference on Information Retrieval (ECIR '23)*, 2023.

Chapter 2

Experimentation Framework

In order to standardise the evaluation of algorithms in our experiments, we have designed a python package `RecPack`*, focused on implicit feedback recommender systems. This package is designed to be easy to use thanks to an interface similar to the popular* `scikit-learn` *python package, and extendible, by making our experiments modular. Each module in the package is linked to a part of an experimental pipeline. In this chapter, we present each of the modules in detail and describe which choices we made in the experiments of this thesis. These choices were inspired by our experiments at Froomle, and our efforts to align them to our online use cases.* [1]

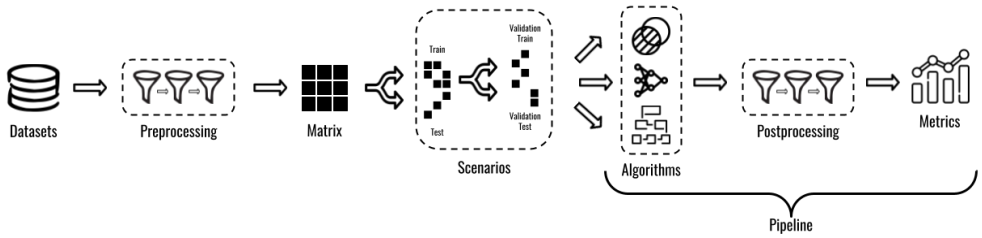

Figure 2.1: Top-N Recommendation Pipeline: First, a dataset is preprocessed and transformed into a user-item interaction matrix. Next, this matrix is split into a training, validation and test dataset. These datasets are then used to first train algorithms and later make recommendations. Finally recommendations are postprocessed, after which performance metrics are computed.

[1] Chapter based on "RecPack: An(other) Experimentation Toolkit for Top-N Recommendation using Implicit Feedback Data". Lien Michiels, Robin Verachtert and Bart Goethals. In Proceedings of the 16th ACM Conference on Recommender Systems [91].

2.1 Introduction

Offline experimentation is an important step in evaluating recommender systems. For many researchers, it is the only tool they have available to evaluate new algorithms they or others have developed, as they do not have access to online platforms. Even for those that do have access to an online platform, the business may not be happy to expose any new algorithm or system to their users. Besides the potential for poor recommendations, there is also the engineering cost to be considered, as it is much harder to put code into production than it is to write something that works for you on a curated dataset.

This indicates the need for efficient and correct evaluation of recommender systems in offline experiments. In recent years, other Python packages for top-N recommendation have been released [e.g. 7, 123, 142]. However, these focus more on the purpose of 'benchmarking', i.e., quick and accurate comparisons of state-of-the-art recommendation algorithms. Consequently, they provide access through the use of a configuration language or command-line interface. `RecPack` on the other hand wishes to support researchers with the development of their own algorithms through repeated refinement, experimentation and bug-fixing in e.g., a Notebook environment.

Furthermore, most of the algorithms implemented in `RecPack` allow induction of recommendations for new users (unseen during training) as well as more accurate recommendations by using new data, that was not yet available at training time. Most of the other packages on the other hand require a user to be known during training time, and can only transduce recommendations based on the training data.

As shown in Figure 2.1, `RecPack` is made up of seven main components, which all contribute to the final result of the experiment. We will introduce each of the components, showcase the use of the package through the pipeline, and end with suggestions for setting up offline experiments to mimic online use cases.

2.2 Datasets

In offline experiments, we assume to have access to a static dataset $\mathcal{D}$, which contains interactions of users with items. These can be publicly available datasets, such as MovieLens[50] or Adressa[46], or exports from industry partners. When setting up our experiments, we require that the dataset contains timestamp information for each of the events. Any production setting will have access to this information, and most recommender systems will actively use it. As we will explain in Section 2.4, these timestamps will allow us to mimic a more realistic evaluation, even if the algorithms might not use them.

Choosing the right dataset for an experiment is an important first step to getting accurate and usable results. When dealing with production settings at Froomle, we are fortunate to have access to the datasets of the websites for which we aim to improve personalisation. Choosing the right publicly available dataset then, is usually a search limited by availability. Constraints on type of data and the availability of timestamps narrow the amount of available datasets. Thankfully, recent years have seen more public datasets

published, allowing us to present our research on these datasets as well as our private datasets.

`RecPack` implements classes for many of these publicly available datasets, and also makes it easy to extend the functionality to load a private dataset. When loading a dataset, by default we apply common preprocessing patterns, which will be further discussed in the next section.

In Table 2.1 we present the public datasets used in this work, and their statistics, both before and after their default preprocessing.

2.3 Preprocessing

Most often the raw dataset is not yet ready for use in experiments. Many users may have only few interactions, some items are visited only a handful of times, or the dataset still contains bot users that are interacting with every item on the website. These factors will influence the evaluation, and make it less accurate. We would like the dataset to suit the task we evaluate, namely "How well does the algorithm personalise recommendations". Evaluating that on users with too few interactions, will skew results towards methods with mostlty popularity bias, as that is the best bet in absence of information. Similarly, items with few visits are only relevant, if you evaluate a cold-start approach, but in most other cases, they will only slow down training, and potentially introduce incoherrent noise into the data. It is therefore common practice to process the raw data into a specific subset of the data, that is more suited for use in experiments evaluating personalised recommendation algorithms [4, 35, 123].

Another type of preprocesing is used to construct an implicit feedback dataset from an explicit dataset. This is necessary as some of the most popular datasets used in recommendation research are explicit feedback datasets (e.g. MovieLens [50] and Amazon review [93] datasets). It is common practice to either use each rating as an implicit feedback, or to keep only those interactions with a rating above 4 as the positive implicit feedback [67, 80].

`RecPack` provides the following preprocessing filters that are frequently applied in processing a dataset:

`Deduplicate` Removes interactions between users and items such that every (user, item) pair occurs at most once. Keeps the last interaction if timestamps are available.

`MinRating(r)` Removes all ratings from the dataset below the specified threshold (`r`). This filter is used to translate an explicit feedback dataset into an implicit feedback dataset by dropping interactions that have a rating below a threshold (typically 3-stars or 4-stars), leaving us with only a subset of interactions for items the user rated highly.

MinItemsPerUser(c) Removes all interactions from users that have too few interactions (less than c). Since we are using the dataset for evaluating personalisation, users with too few interactions are not representative of our target audience, and so using them will skew results.

MinUsersPerItem(c) Removes all interactions with items that have too few interactions (less than c). Not all items are visited equally, and some are only visited a handful of times in a dataset with millions of interactions. Because many algorithms have their complexity and performance scale with the number of items, it makes sense to remove these extremely unpopular items. Further, production environments may have similar filters in place, to avoid recommending potentially bad recommendations to users. Only in specific discovery modes, such as cold start scenarios will they allow rarely visited items to be shown.

MaxItemsPerUser(c) Removes users that have interacted with too many items (more than c). These users usually are bots visiting most of the catalogue. These are obviously not the target audience for personalisation, and their behaviour can work detrimental to models and evaluation because they visit disproportionate amounts of items compared to regular users.

NMostRecent(n) Selects only events on the n most recently visited items. Sometimes datasets contain so many items, that algorithms require too much memory, or run time is too high to allow quick experimentation and development of algorithms. In these cases it is better to initially work on a smaller subset of the dataset, by reducing the amount of items used in the experiment.

NMostPopular(n) Selects only interactions with the n most popular items, this is another way of drastically shrinking the dataset for quick iteration purposes.

Dataset	Duration	Processed	$\lvert U \rvert$	$\lvert I \rvert$	$\lvert \mathcal{D} \rvert$	ρ (%)
Adressa [46]	7 days	No	640 503	20 428	3 101 991	0.02
		Yes	228 462	2 790	2 532 729	0.40
Amazon Games [93]	6 928 days	No	55 223	17 408	497 577	0.05
		Yes	42 480	14 817	385 543	0.06
Amazon Toys & Games [93]	6 939 days	No	208 180	78 772	1 828 971	0.01
		Yes	172 606	69 716	1 531 360	0.01
CosmeticsShop [69]	152 days	No	1 597 754	53 854	9 657 821	0.01
		Yes	483 080	27 019	7 877 677	0.06
Globo.com [31]	44 days	No	322 897	46 033	2 988 181	0.02
		Yes	218 228	9 759	2 722 355	0.12
MovieLens1M [50]	1 040 days	No	6 040	3 706	1 000 209	4.47
		Yes	6 038	3 125	574 385	3.04
MovieLens25M [50]	9 082 days	No	162 541	59 047	25 000 095	0.26
		Yes	162 342	19 937	12 416 034	0.38
Yoochoose [17]	182 days	No	9 249 729	52 739	33 003 944	0.01
		Yes	9 244 493	37 964	32 972 850	0.01

Table 2.1: Public datasets used in this work and their stats computed both before and after applying the preprocessing filters. ρ is the density of the dataset.

2.4 Scenarios

Once the dataset is preprocessed, the next step is to construct training, validation and test datasets. We call this split, a "scenario". Different scenarios create different problems and thus different evaluation results.

Each scenario constructs the following six sub-datasets.

- Validation training data ($\mathcal{D}_{v\_train}$): The dataset on which models are trained during hyperparameter optimisation.
- Validation input data ($\mathcal{D}_{v\_in}$): The interaction history of users that will be used during the prediction (for hyper parameter optimisation). For some scenarios this is identical to $\mathcal{D}_{v\_train}$.
- Validation output data ($\mathcal{D}_{v\_out}$): The targets to predict for each user. It can be one item per user or many, depending on the scenario used.
- Full training data ($\mathcal{D}_{train}$): The dataset on which models are trained before final evaluation (with optimised hyperparameters).
- Test input data ($\mathcal{D}_{in}$): The interaction history of users that will be used during prediction.
- Test output data ($\mathcal{D}_{out}$): The targets to predict for each user. Can be one item per user, or many.

When splitting data, we always make sure that each user with an interaction in $\mathcal{D}_{in}$ also has at least one interaction in $\mathcal{D}_{out}$, and vice versa. This way we make sure that all active users (users with an interaction in $\mathcal{D}_{in}$) have a target item, and we don't have target items that need to be predicted without previous interaction. Investigating non-personalised recommendations is not the goal of these scenarios.

Further, the design of our scenarios makes sure that the validation and testing settings are as closely aligned as possible, by making sure they work on similar amounts of data.

In `RecPack`, we have implemented a selection of 7 scenarios that perform different splits of the dataset into these sub-datasets. These are categorized in Table 2.2, according to three properties: "Do they perform a hard timed split?", "Do they perform a sequence aware evaluation?" and "Do they avoid overlap between users used for training and evaluation?".

The `StrongGeneralization` scenarios provide an extra difficulty to the algorithms, by completely separating users used for training and evaluation. For algorithms that rely on building user features or embeddings during training, these scenarios are not suitable, because history of users to predict for, will only become available at prediction time.

In the following, we will describe each of the seven implemented scenarios in detail, and link them to real world use-cases with which they align.

Timed Split	Sequence Aware: No	Sequence Aware: Yes
No	`WeakGeneralization`	`LastItem`
No	`StrongGeneralization`	
Yes	`Timed`	`TimedLastItem`
Yes	`StrongGeneralizationTimed`	`StrongGeneralizationTimedLastItem`

Table 2.2: Categorisation of scenarios according to their properties. Strong generalization scenarios, which do not reuse users for training, validation and testing, are highlighted in gray. A scenario could be designed that fills the gap of a non-timed, strong-generalization, last-item scenario. However, we have not found any papers using this method, and it would suffer from similar leakage issues as the `LastItem` scenario. Therefore, we choose to not implement it in RecPack.

2.4.1 Weak Generalization

In this scenario, the data is split per user by performing two splits of $\mathcal{D}$, based on a single parameter f_{in} defining the fraction of data per user that is split into a subset "in" and the remainder in "out".

The first split's "in" dataset is used as both $\mathcal{D}_{train}$ and $\mathcal{D}_{in}$, the "out" dataset is used as $\mathcal{D}_{out}$. The second split further divides the $\mathcal{D}_{train}$ dataset using the same f_{in} ratio as the first for validation purposes. Now the "in" part is used for $\mathcal{D}_{v\_train}$ and $\mathcal{D}_{v\_in}$, while the "out" part is used for $\mathcal{D}_{v\_out}$.

When dividing a user's history into the "in" dataset, we round up, this can cause some users to have no items in the "out" side of the split, which removes them from evaluation datasets, but not from training.

An example of splitting a dataset using the `WeakGeneralization` scenario is given in Figure 2.2.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(a) Full training dataset

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(b) Validation Training dataset

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(c) Validation input dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(d) Validation target dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(e) Test input dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(f) Test target dataset.

Figure 2.2: Resulting training, validation and testing matrices for the `WeakGeneralization` scenario, with $f_{in} = \frac{1}{2}$. Columns are timestamps, rows are user histories, each non-empty entry is an item.

2.4.2 Last Item

Users consume items in sequential order and sequence-based recommendations seek to exploit this sequential nature of the data to provide better recommendations [13, 67, 84]. In order to evaluate these algorithms, the scenario needs to be designed in such a way that the sequential nature of interactions is maintained, and represented in the prediction task. When splitting the data, each user's last interaction $\mathcal{D}_u^l$ is used as test target ($\mathcal{D}_{out} = \{\mathcal{D}_u^l \, \forall \, u \in U\}$) and the second to last $\mathcal{D}_u^{ll}$ is used as validation target ($\mathcal{D}_{v\_out} = \{\mathcal{D}_u^{ll} \, \forall \, u \in U\}$).

Training data is made up of all events before the target event: $\mathcal{D}_{train} = \mathcal{D} \setminus \mathcal{D}_{out}$ and $\mathcal{D}_{v\_train} = \mathcal{D}_{train} \setminus \mathcal{D}_{v\_out}$.

In some situations, it is beneficial to not use all the historic events from a user to predict the target item, but instead use only a subset. Thus, we allow the reduction of the input data for prediction, to the n most recent interactions. In traditional sequential recommendation evaluation, n is set to infinity, and so all events used for training are also used for prediction. In production settings, it is often opportune to set this parameter to 1, only using the last seen item to personalise recommendations. While this throws away most of the knowledge about the user's interests, it captures the current interest, and drastically reduces response times, which is an important factor when deploying an algorithm in production.

An example of the scenario applied to a sample dataset can be found in Figure 2.3.

Ji et al. [65] have criticised this scenario for leaking future information into training data. This leakage comes from the fact that training data is created per user, such that a user with most of their interactions on one of the last days of the dataset, will contribute interactions to the training data from those days, while the trained model, will also be evaluated on its ability to predict items for users that were active weeks before those interactions. This causes future knowledge to be encoded into the model, something that in production will never be available. Despite this, we support the `LastItem` scenario, as it is still commonly used in research [105, 27, 68, 77]. We do recommend considering using the `TimedLastItem` variant over this one though.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(a) Full training dataset

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(b) Validation Training dataset

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(c) Validation input dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(d) Validation target dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(e) Test input dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(f) Test target dataset.

Figure 2.3: Resulting training, validation and testing matrices for the `LastItem` scenario, with $n = 1$. Columns are timestamps, rows are user histories, each non-empty entry is an item.

2.4.3 Timed

In production settings, models are trained on data available up to the moment of training. They are also only used to predict user behaviour until a new model is trained.

We mimic this online behaviour in this scenario. The scenario is parameterised by two timestamps: T, and T_{val} ($< T$), and two time windows: δ_{in} and δ_{out}. First the data is split on timestamp T, data before T is used for training and as test input, data after T as test targets. Then δ_{out} limits the test targets to only events within that many seconds of T and δ_{in} limits both training and test input data to only recent interactions. To create validation data, the same procedure is used, only now surrounding T_{val}.

Formally:

- $\mathcal{D}_{train} = \{(u, i, t) \in \mathcal{D} \mid t >= T - \delta_{in} \wedge t < T\}$
- $\mathcal{D}_{in} = \{(u, i, t) \in \mathcal{D} \mid t >= T - \delta_{in} \wedge t < T\}$
- $\mathcal{D}_{out} = \{(u, i, t) \in \mathcal{D} \mid t >= T \wedge t < T + \delta_{out}\}$
- $\mathcal{D}_{v\_train} = \{(u, i, t) \in \mathcal{D} \mid t >= T_{val} - \delta_{in} \wedge t < T_{val}\}$
- $\mathcal{D}_{v\_in} = \{(u, i, t) \in \mathcal{D} \mid t >= T_{val} - \delta_{in} \wedge t < T_{val}\}$
- $\mathcal{D}_{v\_out} = \{(u, i, t) \in \mathcal{D} \mid t >= T_{val} \wedge t < T_{val} + \delta_{out} \wedge t < T\}$

The single time-based split has been used in multiple works to evaluate recommendation algorithms taking into account the dynamic nature of data [24, 98, 127]. Most commonly in these works, δ_{in} and δ_{out} are set to infinity, thus using all data.

Inspired by sliding-window evaluation [61] and retraining in production, we make use of δ_{out} to limit the target data. A model usually does not have to predict events more than a few hours or days into the future. The importance of δ_{in}, is further discussed in Chapter 4, where we analyse why forgetting some events actually helps improve recommendation quality.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(a) Full training dataset

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(b) Validation Training dataset

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(c) Validation input dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(d) Validation target dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(e) Test input dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(f) Test target dataset.

Figure 2.4: Resulting training, validation and testing matrices for the `Timed` scenario, with $T = 6$, $T_{val} = 4$ and $\delta_{in} = \delta_{out} = \infty$. Columns are timestamps, rows are user histories, each non-empty entry is an item.

2.4.4 Timed Last Item

In Section 2.4.2, we presented the `LastItem` scenario, which suffered from leakage as shown by Ji et al. [65].

In this scenario, we address the problem of leaking information from the future, while still maintaining a sequential evaluation.

We follow the temporal divisions as introduced in the `Timed` scenario before. Only data before a timestamp T is used for training. $\mathcal{D}_{train} = \{(u, i, t) \in \mathcal{D} \mid t < T \wedge t >= T - \delta_{in}\}$. To create the test dataset we extract only users that have at least 1 event in the interval $[T, T + \delta_{out}[$ and use all but their last interaction (also those before T) as history to predict this last interaction, as in the `LastItem` scenario.

To tune hyperparameters, we introduce a second cut-off timestamp $T_{val} < T$, such that our training dataset during hyperparameter optimisation are the events $\{(u, i, t) \in \mathcal{D} \mid t >= T_{val} - \delta_{in} \wedge t < T_{val}\}$. To obtain the validation targets, we extract users that have an interaction in the interval $[T_{val}, \min(T, T_{val} + \delta_{out})[$ and, analogous to the test dataset, use all prior interactions to predict these users' final interaction. Further, the parameter n defines how much of the prior interactions will be used as history for the user when requesting recommendations, analogous to the `LastItem` scenario.

An example of splitting a dataset can be found in Figure 2.5

2.4.5 Strong Generalization

In all of the previous scenarios, interactions from the same user were used for training, validation and testing. One consideration to make, is that we would like our algorithms to generalize the knowledge they learn.

One way to evaluate this, is to make sure that users are never used in more than one task. If an event from a user is used for training, none of that user's events should be used for evaluation. This is commonly referred to as a "Strong Generalization" split.

Thus in this scenario, the scenario first divides the set of users U into two disjunct subsets U_{train}, U_{test} and then further splits U_{train} to get validation training (U_{v_train}) and evaluation users (U_{val}). We construct these user sets by randomly assigning each user to one of the groups according to the parameter f_{train}, the fraction of users to use for training. Thus, $|U_{train}| = f_{train} \times |U|$ and $|U_{test}| = (1 - f_{train}) \times |U|$, and for validation data, $|U_{v\_train}| = f_{train} \times |U_{train}|$ and $|U_{val}| = (1 - f_{train}) \times |U_{train}|$.

The training datasets $\mathcal{D}_{train}$ and $\mathcal{D}_{v\_train}$ contain all events of their respective user sets. The evaluation users' data is split similarly to the `WeakGeneralization` scenario, f_{in} interactions are used as input data, while the remainder is used as target data.

Because users are not shared between training and evaluation, some algorithms are incompatible with this scenario. Factorization methods such as SVD [49] and BPRMF [107] learn a user embedding during training and use it instead of the interaction history to generate predictions. When users do not overlap between training and evaluation, they will predict randomly, or give no recommendations.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(a) Full training dataset

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(b) Validation Training dataset

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(c) Validation input dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(d) Validation target dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(e) Test input dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(f) Test target dataset.

Figure 2.5: Resulting training, validation and testing matrices for the `TimedLastItem` scenario, with $T = 6$, $T_{val} = 4$, $\delta_{in} = \delta_{out} = \infty$ and $n = 1$ Columns are timestamps, rows are user histories, each non-empty entry is an item.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(a) Full training dataset

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(b) Validation Training dataset

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(c) Validation input dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(d) Validation target dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(e) Test input dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(f) Test target dataset.

Figure 2.6: Resulting training, validation and testing matrices for the `StrongGeneralization` scenario, with $f_{train} = \frac{2}{3}$ and $f_{in} = \frac{1}{2}$. Columns are timestamps, rows are user histories, each non-empty entry is an item.

2.4.6 Strong Generalization Timed

This adaption of the `StrongGeneralization` scenario follows the same ideology: "never share users between training and evaluation", and also incorporates the temporal aspect of recommendation.

Users are still split into distinct sets as in `StrongGeneralization`. For the training dataset though, we now only use the interactions before a timestamp T (and within δ_{in} seconds of T): $\mathcal{D}_{train} = \{(u, i, t) \in \mathcal{D} \mid u \in U_{train} \wedge t < T \wedge t >= T - \delta_{in}\}$.
Similarly, for the validation training data, we use a second timestamp T_{val} ($T_{val} < t$): $\mathcal{D}_{v\_train} = \{(u, i, t) \in \mathcal{D} \mid u \in U_{v\_train} \wedge t < T_{val} \wedge t >= T_{val} - \delta_{in}\}$.

For the evaluation datasets we also apply a timed split with timestamps t and t_{val}, rather than the random split used before:

- $\mathcal{D}_{in} = \{(u, i, t) \in \mathcal{D} \mid u \in U_{test} \wedge t < T \wedge t >= T - \delta_{in}\}$
- $\mathcal{D}_{out} = \{(u, i, t) \in \mathcal{D} \mid u \in U_{test} \wedge t >= T \wedge t < T + \delta_{out}\}$
- $\mathcal{D}_{v\_in} = \{(u, i, t) \in \mathcal{D} \mid u\ in U_{val} \wedge t < T_{val} \wedge t >= T_{val} - \delta_{in}\}$
- $\mathcal{D}_{v\_out} = \{(u, i, t) \in \mathcal{D} \mid u \in U_{val} \wedge t >= T_{val} \wedge t < T_{val} + \delta_{out}\}$

An example of this scenario can be found in Figure 2.7.

2.4.7 Strong Generalization Timed Last Item

In analogy to the `StrongGeneralizationTimed` scenario, this is the variant of the `TimedLastItem` scenario, with users not shared between training, validation and test sub-datasets.

Again, users are split into distinct sets as in `StrongGeneralization`. Following the `Timed` scenario, the training dataset is limited to interactions before a timestamp T (and within δ_{in} seconds of t): $\mathcal{D}_{train} = \{(u, i, t) \in \mathcal{D} \mid u \in U_{train} \wedge t < T \wedge t >= T - \delta_{in}\}$. For the validation training data, we also use a second timestamp $T_{val} < T$, such that $\mathcal{D}_{v\_train} = \{(u, i, t) \in \mathcal{D} \mid u \in U_{v\_train} \wedge t < T_{val} \wedge t >= T_{val} - \delta_{in}\}$.

For the evaluation datasets, we follow the `LastItem` scenarios, and split the last item as target, and use the n most recent items as input.

An example of the scenario is worked out in Figure 2.8

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(a) Full training dataset

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(b) Validation Training dataset

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(c) Validation input dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(d) Validation target dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(e) Test input dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(f) Test target dataset.

Figure 2.7: Resulting training, validation and testing matrices for the `StrongGeneralizationTimed` scenario, with $T = 6$ $T_{val} = 4$ and $\delta_{in} = \delta_{out} = \infty$. Columns are timestamps, rows are user histories, each non-empty entry is an item.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(a) Full training dataset

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(b) Validation Training dataset

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(c) Validation input dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(d) Validation target dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(e) Test input dataset.

t	1	2	3	4	5	6	7
u_1	1	2				3	
u_2		3	4	5	2		
u_3	5	3		2		1	4
u_4			1		3		
u_5		5	2	1		4	
u_6					3	2	1

(f) Test target dataset.

Figure 2.8: Resulting training, validation and testing matrices for the `StrongGeneralizationTimedLastItem` scenario, with $T = 6$, $T_{val} = 4$, $\delta_{in} = \delta_{out} = \infty$ and $n = 1$. Columns are timestamps, rows are user histories, each non-empty entry is an item.

2.5 Algorithms

The algorithm is the main variable we want to evaluate when performing offline evaluations. Many algorithms have been proposed, each solving a recommendation problem in their own way.

`RecPack` provides a rich selection of state-of-the-art algorithms, covering some of the most popular recommendation paradigms: Item-to-item similarity [32, 44, 95, 128], factorization [4, 54, 107], auto-encoders [80, 116, 121], session-based [42, 53] and time-aware recommendation algorithms [81].

Before discussing the algorithms used in this thesis in more detail, we list all algorithms currently implemented in `RecPack`.

- Popularity
- Random
- SLIM [95]
- ItemKNN [32]
- TARSItemKNNLiu [81]
- NMFItemToItem [39]
- SVDItemToItem [49]
- Prod2Vec [44]
- Prod2VecClustered [44]
- KUNN [128]
- NMF [39]
- SVD [49]
- WeightedMatrixFactorization [54]
- BPRMF [107]
- RecVAE [116]
- MultVAE [80]
- EASE^{R} [121]
- GRU4Rec [53]
- STAN [42]

Popularity Recommends items based on how frequently they have been interacted with in the training dataset. While this is not a personalised algorithm, it is often good baseline.

Item-kNN One of the most well-known and frequently used baseline algorithms for neighbourhood-based collaborative filtering [32, 113]. The model consists of a single matrix multiplication with the item-item similarity 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 this thesis we use either cosine similarity or the conditional probability inspired similarity defined in Deshpande and Karypis [32]. Recent work on news recommendation highlights the remarkable competitiveness of simple neighbourhood-based methods compared to more complex alternatives [85, 92].

EASE$^{\mathrm{R}}$ Auto encoder, without dimensionality reduction. The model was proposed as an extension of the well-known SLIM method [95, 121]. In EASE$^{\mathrm{R}}$, the item-item matrix $\boldsymbol{S}$ is learned through a least-squares optimisation problem that allows for a closed-form solution and optimises the auto-encoder problem $|\boldsymbol{X}\boldsymbol{S} - \boldsymbol{X}|_2$. This closed-form solution makes the model 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$^{\mathrm{R}}$ 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 [134]. 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 Bayesian Personalised Ranking (BPR) loss to optimise the model, rather than using cross-entropy loss. BPR is more suited for our evaluations, 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) Generates sequential association rules between items. The model recommends items related to the last item a user has seen: $\phi(\boldsymbol{X}) = \boldsymbol{X}^l\boldsymbol{S}$. Where $\boldsymbol{X}^l$ is the binary last visit matrix, $\boldsymbol{X}^l_{u,i} = 1$ only if i is the last item visited by user u. The asymmetric similarity $\boldsymbol{S}_{i,j}$ between items i and j, is computed as $\sum_{u\in U} \frac{\mathbb{I}(u,i,j)}{\mathrm{gap}(u,i,j)}$. Where $\mathbb{I}(u,i,j)$ 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 a co-occurrence is ignored. Ludewig et al. [86] found that despite the simplicity of the algorithm, it performed competitively in sequential recommendation tasks.

Item-KNN with time-decay While the traditional Item-KNN algorithm assumes a binary implicit feedback, this group of variants weight the implicit feedback, by decaying the value depending on how long ago each interaction occured. Older interactions get lower values to indicate less relevance to the model. This non-binary matrix is then used to compute the similarity matrix $\boldsymbol{S}$ using various similarity functions. Cosine similarity, Pearson-correlation and Conditional Probability Inspired similarities are supported. Methods that apply this ideas have been proposed by different author [33, 81, 83, 126, 135]. Each of them proposing different combinations of similarity functions and decay functions. In Chapter 5 we present a review of the literature, and generalise the different methods into a framework.

Item-KNN with time between events decayed This group of methods defines specialised similarity computations between items, given timestamps are available for occurring events [135, 51]. Rather than decaying each event's score individually, and then computing a traditional similarity between two vectors, these methods incorporate the temporal dynamics into the similarity computation. The core idea is that interactions occurring closely together in time, are more indicative of a relationship between two items, compared to interactions that occurred further apart. Intuitively, when a user visits items A and B within the hour, and then visits C the next day, items A and B are more related than A and C or B and C. In Chapter 5 we review the literature, and fit these methods into a holistic Time-aware Item-KNN framework.

MultVAE Neural Network proposed by Liang et al. [80], which uses Variational Auto-Encoders (VAEs) for Collaborative Filtering.
VAEs approach the recommendation problem as a reconstruction problem, and learn a function ϕ such that $\phi(X) \approx X$. In this case the function is a Neural Network, which uses a bottleneck layer to force the function to generalise. The parameters of the function are optimised using Gradient Descent with a combination of the Binary Cross Entropy loss and Kullback–Leibler Divergence as loss function.

BPR-MF Factorizes the implicit feedback matrix X into two lower rank matrices $W \in \mathbb{R}^{|U| \times k}$ and $H \in \mathbb{R}^{|I| \times k}$ such that $WH^T \approx X$, where k is a hyperparameter. Instead of minimising the reconstruction loss ($||WH^T - X||_2$) as done in traditional matrix factorization, this method instead optimises a Bayesian Personalised Ranking (BPR) loss [107]. This loss takes into account that zeros in the original matrix are not explicit zeros, and so it is more important for the reconstructed matrix to score seen items (known positives) higher than unseen items (unknown zeros), than it is for the model to predict exactly 1 or 0 for the seen and unseen items respectively .

2.6 Postprocessing

Real world recommender systems often apply `postprocessing`, in the form of business rules, to the predictions made by the recommendation algorithm. In an e-commerce context, for example, it is customary to exclude sensitive or age-restricted items. In a news context, the recommendations could be limited to articles published during the last day.

In `RecPack` we implemented two commonly used filters, as well as provide a baseclass to allow users to define any postprocessing step they would like.

`SelectItems` Allows only recommendations from a specific subset of the catalog. This can be useful to select recent items, in scenarios where recency is important, or items from a specific category in case of recommendations in a very specific context.

`ExcludeItems` Removes items that we don't want to recommend. Typical examples include sensitive or age-restricted items in a retail context or click-bait news in a serious newsletter.

2.7 Metrics

Of course, the ultimate goal of a recommendation experiment is to evaluate the performance of a recommendation algorithm. `RecPack` comes with a selection of the most commonly used metrics in Top-K ranking [4, 108].

We purposefully designed `RecPack`'s metrics to allow for a fine-grained analysis of the performance of algorithms, as well as giving a more traditional single-value metric. To achieve this, the metrics store detailed information (such as hits per user), which can be used to aggregate over subgroups of the user base instead of over the entire user base. This can be used to analyse the metrics depending on the activity level of the user, or some additional user features if available [97, 118].

In Table 2.3 we present the list of metrics implemented in `RecPack` and highlight how much detail they store.

Choosing the right metric is an important consideration when setting up an experiment. Often metrics will have contradicting results. For example, the most accurate algorithm, might not be the one covering most of the catalogue, or the algorithm with the highest recall is most likely not the best ranker, and so might have a worse NDCG or MRR value.

It is often an art to find the metric that aligns the best with the online business metrics (CTR, fairness, revenue, ...). For each of our experiments, we made specific choices for metrics, depending on the needs of Froomle's customers. Most often, we chose to use NDCG as a quality metric since we found it to align the best when CTR is the optimisation target. Many of the real world use-cases present items in a vertical list, which means we value good recommendations on higher positions more. We also use Recall, as

Metric	Global	User	User-Item
CoverageK	✓		
PercentileRankingK	✓		
DCGK		✓	
NDCGK		✓	
RecallK		✓	
CalibratedRecallK		✓	
PrecisionK		✓	
ReciprocalRankK		✓	
HitK			✓
DiscountedGainK			✓

Table 2.3: The list of metrics implemented in `RecPack`, and how detailed the results can be inspected. Global metrics only store a single value for each evaluation. User metrics store intermediate results per user, a single value is obtained by averaging these across all users. Finally, the user-item based metrics store a value per user-item pair in the top-k recommendations (hit or no hit, or a discounted value based on the rank of the hit), the overall value is obtained by aggregating all of these scores.

sometimes lists are presented horizontally, or the position bias is unknown for a new use case. Further, coverage of the item catalogue is an important fairness metric, to convince business stakeholders that they can benefit from using personalisation. We discuss in more detail each of these metrics used in this thesis.

Normalised Discounted Cumulative Gain (NDCG) Ranking metric, computed on a ranked top-k list, that gives lower values to hits lower down in the ranking of recommendations.

To compute, we need to compute Discounted Cumulative Gain (DCG) and the ideal DCG (iDCG). Given a ranked list of k recommendations (r_u) for a user, and a target matrix Y, where $Y_{u,i} = 1$ if the user-item interaction (u, i) is part of the target dataset:

$$\text{DCG}(u) = \sum_{i \in r_u} \frac{Y_{u,i}}{\log_2(\text{rank}(r_u, i) + 1)}$$

$$\text{iDCG}(u) = \sum_{j=1}^{\min(K, |Y_u|)} \frac{1}{\log_2(j+1)}$$

Where rank() is a function that returns the rank of recommended item i in recommendation list r_u and $|Y_u|$ are the number of targets in Y for user u. To compute the NDCG, we compute

$$\text{NDCG} = \frac{1}{|U|} \sum_{u \in U} \frac{\text{DCG}(u)}{\text{iDCG}(u)}$$

The advantage of normalizing the DCG scores, is that it helps prevent users with more targets to dominate the final score. Now each user's score is proportional to the maximal score they could have scored instead.

(Mean) Reciprocal Rank (MRR) Ranking metric, computed on a ranked top-k list, which only counts the first hit in computing the metric value. Reciprocal rank uses a different value-lowering score than NDCG, a hit at position n gets a score of $\frac{1}{n}$. The Reciprocal Rank (RR) for a user is defined as follows:

$$\mathrm{RR}(u) = \max_{i \in r_u} \frac{Y_{u,i}}{\mathrm{rank}(r_u, i)}$$

The mean reciprocal rank then, is the average of the Reciprocal Rank over all users.

Calibrated Recall Accuracy metric, which computes which portion of the targets have been recovered for each user by the recommender system. We use the calibrated variant proposed by Steck [121], to avoid unfair low scores for users with more than k targets when evaluating a top-k list (r_u).

$$\mathrm{CalibratedRecall}(u) = \frac{\sum_{i \in r_u} Y_{u,i}}{\min(|Y_u|, k)}$$

A global value is computed by taking the average over each user.

Coverage Fairness metric that is used to measure which fraction of the catalogue is recommended by an algorithm. High coverage is usually a desirable feature for production algorithms, as otherwise many items get avoided by the algorithm.

$$\mathrm{coverage}(u) = \frac{|\{i \in I | (\exists u \in U)[i \in r_u]\}|}{|I|}$$

2.8 Pipelines

Bringing it all together, `RecPack` provides a `PipelineBuilder` class, which helps users to perform the different steps of experimentation in the right order. It allows them to focus on the choices and the experiment design, rather than the implementation details and consistency.

A pipeline requires a chosen scenario applied to a dataset, a set of algorithms to evaluate and their hyperparameter spaces to explore during optimisation, a set of metrics to evaluate and a single optimisation metric, and optionally post-processing steps.

When run, the pipeline sequentially optimises and evaluates each algorithm, taking care to that each result is comparable between algorithms, finally storing the results for analysis.

In the example below, we run the pipeline for a selection of algorithms on the Adressa Dataset.

```
import datetime
from recpack.algorithms import BPRMF, ItemKNN, EASE
from recpack.datasets import Adressa
from recpack.pipelines import PipelineBuilder
from recpack.scenarios import Timed


# Load Dataset
dataset = Adressa()
im = dataset.load()

# Split data into training, validation and test dataset
scenario = Timed(
    t=int(datetime.datetime(2017, 1, 7, 12).strftime("\%s")),
    t_val=int(datetime.datetime(2017, 1, 6, 12).strftime("\%s")),
    delta_out=12*3600,
    validation=True
)
scenario.split(im)

# Build the experimentation pipeline
builder = PipelineBuilder()
builder.set_data_from_scenario(scenario)

builder.add_algorithm(
    algorithm = 'ItemKNN',
    grid = {'K': [50, 100, 200]}
)
builder.add_algorithm(
    algorithm = 'EASE',
    grid = {'l2': [10, 100, 1000]}
)
builder.add_algorithm(
    algorithm = 'BPRMF',
    grid={
        'num_components': [50,100,150],
        'lambda_h': [0, 0.001, 0.1],
        'lambda_w': [0, 0.00, 0.1]
    }
)

builder.add_metric('NDCGK', K=10)
builder.add_metric('CoverageK', K=10)

builder.set_optimisation_metric('NDCGK', K=10)

pipeline = builder.build()

# Run the pipeline
pipeline.run()

# Get the results
pipeline.get_metrics(short=True)
```

2.9 Aligning experiments with production tasks

While offline experiments are rarely an exact oracle of online performance, researchers and practitioners should be able to trust them to make valid comparisons, that allow us to identify interesting options to try in production.

To be able to do this, we need to make sure that the offline experiments are similar in setup to online experiments. This impacts the choices of preprocessing, scenario and metrics we choose when setting up the experiment. In the following paragraphs, we give an overview of the choices made in the experiments of this thesis, and why we make them.

Preprocessing In preprocessing, it is important to match the online preprocessing when running offline experiments. For example, in many of our online model training, we remove users with too few interactions (< 5), because those don't carry as much information as longer histories, but can dominate the similarity computation because there are so many of them. Further, we also remove items that have been visited too infrequently (< 10 or < 50), as they would only inflate the model, without any significant impact on the outcome of the recommendations. A final typical preprocessing we do is to remove users with too many interactions (> 1000), as they are most likely bot users or website auditors, that visit all of the pages, without preference. Using these users can negatively impact models, as they contain no meaningful patterns in their histories.

Scenario The choice of how to split the training, validation and test data is aligned with the context in which the recommendations are used. For example, if the use case is a recommendation list on an article page, suggesting the next articles to read, then we will use the Timed-Last-Item scenario, to incorporate the sequential nature, as well as the temporal aspect, that training data is only available for the past. For lists on home pages, we use the Timed scenario instead, as we might not know the user's intent for the session, but want to allow for a broad spectrum of relevant items if the users visit them in the target period. In all of our experiments, we use scenarios that perform a temporal split for training data, as that aligns with our online retraining policy. When a single evaluation window does not contain enough data points to make a conclusive comparison between algorithms, we use a sliding window approach [61], which gives us more evaluation evidence for different periods.

Metrics In our choice of metrics, we try to find proxies for what we value in our online results. As Click-Through-Rate(CTR) is our primary performance metric in online comparisons, we use NDCG and Recall as a proxy in our offline experiments. NDCG typically is better aligned, because it takes into account the ranking of the recommendations, and we know that users are sensitive to the order in which items were shown. Neither is entirely aligned with the online metrics though, as we'll see in Chapter 6, mostly due to popularity bias in the collected data. An important argument to convince editors to use personalisation is that we can cover more of the items they write than manual selection could. And so we typically measure coverage to see if an algorithm is better at covering the full catalogue. Finally, each online use case has a fixed amount of items (k)

that will be shown to a user, so we make sure to compute our metrics only on the top-k recommendations, as anything below would not be exposed to the user.

2.10 Conclusion

In this chapter, we have presented an in-depth discussion of RecPack, the framework for reproducible and reusable experimentation that we developed. We focus specifically on "scenarios', i.e., the way data is split for offline evaluation. `RecPack` presents a series of novel recommendation scenarios that align with different online recommendation tasks. We presented suggestions for best practices to get offline and online results to align.

Chapter 3

Scheduling on a Budget: Avoiding Stale Recommendations with Timely Updates

Recommendation systems usually create static models from historical data. Due to concept drift and changes in the environment, such models are doomed to become stale, which causes their performance to degrade. In live production environments, models are therefore typically retrained at fixed time-intervals. Of course, every retraining comes at a significant computational cost, making very frequent model updates unrealistic in practice. In some cases, the cost is worth it, but in other cases an update could be redundant and the cost an unnecessary loss. The research question then consists of finding an acceptable update schedule for your recommendation system, given a limited budget. This chapter provides a pragmatic analysis of model staleness for a variety of collaborative filtering algorithms in news and retail domains, where concept drift is a known impediment. We highlight that the rate at which models become stale is highly dependent on the environment they perform in and that this property can be derived from data. These findings are corroborated by empirical observations from four large-scale online experiments. Instead of retraining at regular intervals, we propose an adaptive scheduling method that aims to maximise the accuracy of the recommendations within a fixed resource budget. Offline experiments show that our proposed approach improves recommendation performance while keeping the cost constant. Our findings can guide practitioners to spend their available resources more efficiently.[1]

[1]This chapter is based on our work "Scheduling on a Budget: Avoiding Stale Recommendations with Timely Updates". Robin Verachtert, Olivier Jeunen and Bart Goethals. In *Machine Learning with Applications*, Volume 11, 2023.

3.1 Introduction

Recommendation systems are deployed in production environments where they help users find relevant items in typically large catalogues. In modern-day settings, these users generate millions of interactions every day. This continuous stream of information creates two challenges for recommender systems: 1. the amount of data they need to process calls for efficient algorithms that can keep up with these ever-growing data streams, and 2. the dynamic nature of this data calls for frequent model updates [5, 40]. As time passes, new items become available, others disappear, interests of the global population change, seasons make different items relevant, and interests of single users also change over time. These changes are crucial for the recommendation system to take into account.

Much research has concentrated on developing algorithms to better leverage large amounts of data and achieve greater accuracy. Additional improvements also account for concept drift in the data [24, 73].

Recommendation methods are usually evaluated by splitting the dataset into a static training-, validation-, and test-set [25, 60]. While this is effective for evaluating the quality of the models, it does not address the dynamic nature of the data, due to which even the most accurate algorithms inevitably become stale.

Understanding how model staleness affects recommendation systems for specific environments is necessary to determine how frequently models should be updated [23]. Therefore, we first study how recommender systems are impacted by model staleness as a result of concept drift in the environment. We study both the change that models undergo as new data becomes available and the resulting performance degradation. Our experiments in Figure 3.1 show that model staleness impacts quality very quickly in news domains, due to rapidly changing content and user interests. After just a few hours, the models will recommend mostly irrelevant items. In traditional retail settings, drift is much less pronounced, and as a result, models remain useful for longer periods of time.

The straightforward solution to avoid model staleness is to retrain the model. To the best of our knowledge, the natural question "When should the model be retrained?", remains unanswered in the scientific literature. In this chapter, we attempt to formulate an answer to this often overlooked, yet important question.

A naïve answer is to continuously update the computed models. Once a model is trained and deployed, a new training cycle begins that recomputes a model on the updated dataset. In practice however, such a continuous training cycle incurs a significant cost of computational resources. Therefore, the go-to approach is to schedule model updates at regular intervals.

A more elegant solution is to create *incremental* models [5, 8, 62, 63, 132, 130]. Instead of recomputing the model on the *entire* dataset, these models only process newly collected data and update a previously computed model with this new information. These methods have the advantage that the computational cost for each update is typically much lower than when the model is retrained on the full, potentially huge dataset. Still, the same research question holds, since even for incremental methods, a choice must be made on *when* each update is scheduled. This can be continuous, immediately as new

data arrives, or more typically as the cost of computational resources can be high, in batches after a certain amount of time.

We challenge the implicit assumption made in regular interval scheduling that each update results in a similar increase in accuracy. To illustrate this, imagine a local news website. At night, much fewer interactions occur and usually no new articles become available. As a result, we can safely state that any updates scheduled during the night add very little to the quality of the model. Yet, they are roughly equal in cost to model updates that are being computed at peak traffic hours (e.g. lunchtime) when drifting user interests and new items significantly impact the model. A clear need arises for a scheduling procedure that allows models to be trained at more opportune times. Such a scheduling procedure would increase the online performance of the model over the entire time period, whilst keeping computational costs constant. The day-and-night example serves to illustrate this phenomenon, but it should be clear that a scheduling algorithm that learns from the data is superior.

We posit that the goal of a scheduling procedure is to ensure that each update captures the same amount of new *information*. As such, updates should be scheduled more frequently in periods of high *information gain*, and less frequently in periods of lower gain. All this should happen whilst keeping a fixed budget into account. As we do not know the distribution of high-information periods over the next scheduling period, this adds a non-trivial complexity to the problem.

To overcome these challenges, we propose novel methods for scheduling updates based on summary statistics from previously observed batches of data. Our methods aim to detect *when* model updates benefit the system the most, considering either the informational value of the data or changes in the projected output of the model. We show that these methods improve on the widely used approach with regular intervals. Furthermore, our methods provide a solution to create an optimized schedule for a given budget, which is a common concern for practitioners. So far we intentionally did not mention the specific recommendation method used, as we wish to propose a method independent of the specific algorithm used. After all, even complex systems often rely on basic building blocks that need retraining.

The remainder of this chapter is organized as follows: In Section 3.2, we highlight related work. In Section 3.3, we describe the methods used in the analysis of model staleness in a recommendation context, and describe the scheduling methods. In Section 3.4, the experimental results are presented and analysed. Finally, in Section 3.5, we present how our work provides valuable insights and guidance for practitioners that need to balance the trade-off between keeping a model up to date and the computational cost that comes with it.

3.2 Related Work

Detecting concept drift This is a well studied problem in a variety of applications [20, 40, 72, 103, 133, 136]. Its goal is to detect *when* the underlying distribution of incoming observed data samples changes. Typically, these methods either: 1. measure changes in performance, 2. inspect changing properties of the model, or 3. inspect changing properties of the data [72].

Concept drift detection can be used to decide *when* to update a model, by simply coupling the detection of change with the action of a model update. In an environment where change is rapid, online performance measures, such as click-through-rate or offline metrics, such as precision and recall, are not suitable for such purposes, because they typically show high variance, and detecting changes in such metrics in short intervals is prone to false positives. Depending on the sensitivity of the model it is likely to either wait too long (for the confidence estimate to be smoothed by more samples), or act too soon (as each sudden change in the target value due to variance is considered a detected change in the underlying data).

The alternative of model introspection can provide a way forward in specific use-cases, when the model can be efficiently inspected. However, a generally applicable solution should preferably be model-agnostic. Because of these reasons, in this chapter we focus on the properties of the data, as this seems to be the most interesting avenue to decide on model updates in general yet highly dynamic recommendation scenarios.

Increasing the computational efficiency of model training In virtually all practical recommendation applications, models that train faster are favourable. The reason for this is two-fold: 1. New data can be incorporated more quickly to combat concept drift, and 2. The cost of keeping the model up-to-date is reduced.

Different approaches to improve the computation time of models exist. A first class of methods suggests a trade-off between the accuracy of the model and the computation time. Approximate models explicitly trade the exactness of the model for improved computational complexity [10, 79, 122]. Forgetting mechanisms reduce computational complexity by reducing the amount of data used for training [131]. A second way to reduce computational cost is to adopt models that can be updated incrementally, using only the latest batch of information collected, rather than the complete historical data set. Incremental methods either compute an exact model, which is interchangeable with the traditional non-incremental one, while requiring less time to update [62, 132], or use approximate methods [63, 140]. Although both of these approaches to reduce computational cost study the cost-accuracy trade-off, they still lack a method for scheduling the updates. For example, in the extreme case where an update of the model is triggered with every new transaction, the model would be continuously recomputed. This introduces the high cost of a constantly running resource. More typically a fixed schedule is used, such that models are updated at regular intervals. Alternatively, a fixed window size is used, such that the model will be updated after a fixed number of transactions. In this chapter, we challenge both the assumption that every update is equally valuable and the assumption that every transaction carries the same amount of information.

Model staleness In practical machine learning, model staleness is a fundamental pitfall that must be avoided to achieve satisfactory results consistently over time [23]. In recommendation systems this notion is to the best of our knowledge under-explored. Jambor et al. [56] describe an application of systems control theory to the problem of updating the recommendation system. In their method, they make the assumption that each transaction carries the same information value, which we challenge in this chapter. In addition, the method relies on performance metrics to monitor the system, which introduces large variance when looking at very fine grained control at the granularity of minutes.

Zanardi and Capra [138] present an approach using feed-forward control system theory to decide on updating a model. Based on the number of new users and items that have arrived in the system, it decides when to retrain. This approach manages to avoid waiting for the model to degrade by predicting the degradation based on summary statistics. Contrary to the previous method, this method does not assume a linear relationship between transactions and degradation, instead they assume a relationship between new users and items arriving in the system. However, if known users behave differently, this update strategy will not suggest a model update, and similarly, if no new items arrive, the model will also not be retrained, while users' interests in items might be changing.

Al-Ghossein et al. [5] use an adaptive windowing technique (ADWIN) to decide when to update a Latent Dirichlet Allocation (LDA) topic model in production [20, 21]. Their approach monitors the probability that a word occurs in a text, given the inferred LDA model. When this probability shifts significantly, the older data is discarded, and the model is updated using only the more recent data.

Our work differs from these approaches in that it is independent of a specific modelling technique, handles changing behaviour of users, and additionally takes into account the budget available for scheduling model updates, since that is often the variable that is constrained in a real-world scenario.

3.3 Methodology

Throughout this chapter, we assume the user-item interaction data to be binary and positive only, as this encompasses the most commonly encountered use-cases in practice [129]. A dataset $\mathcal{D}$ consists of user-item-timestamp triplets $(u, i, t) \in U \times I \times \mathbb{N}$. We assume the timestamp to be at the granularity of seconds. User-item interactions can make up various different types of events (e.g. view, add-to-cart, click, purchase, ...), we will refer to them as "events" in general. A recommendation model ϕ maps a user representation to a vector of recommendation scores: $\phi : U \to \mathbb{R}^{|I|}$. For a user-item matrix $\boldsymbol{X} \in \mathbb{R}^{|U| \times |I|}$, $\phi(\boldsymbol{X})$ indicates a row-wise transformation from user histories to recommendation scores. For simplicity and without the loss of generality, we do not consider more involved models that take into account contextual features or the sequence of the user history.

3.3.1 Measuring Model Staleness

We formalise the concept of "staleness" in two ways. First, a model is considered stale if its prediction quality is worse than a more recently updated model. Second, a model can be considered stale if its output is significantly different from that of a more recent model.

Accuracy To measure changes in model quality over time, we calculate how accurate the model still is. Calculating the recall metric for models for consecutive, non-overlapping windows of w seconds on the test data. We will refer to these batches of test data as *slices*. As is typical in temporal evaluation, we split the data on timestamps T_{val} and T_0. The user-item interactions with timestamp $t < T_{\text{val}}$ are used for model training, those with $T_{\text{val}} \leq t < T_0$ for validation, and the remaining interactions with $t \geq T_0$ make up the test set.

We further divide the test data into n slices. This allows us to gain a more fine-grained view of model performance in the test set, since we can now obtain n performance estimates where $T_s = T_0 + (s \cdot \text{w})$ and slice s (0-indexed) consists of data $T_s \leq t < T_{s+1}$. Now, we evaluate the out-of-date model ϕ_0 trained on data up to T_0 and the up-to-date models ϕ_i trained on data up to T_s, and observe the impact of model recency on accuracy. This evaluation method boils down to the SW-EVAL method presented by Jeunen et al. [61], where we additionally evaluate stale models on every test slice.

A decrease in quality as the model grows older is a manifestation of model staleness. The slope of the degradation dictates quickly we need to update the models for them to remain up to date.

Output correlation Two models might attain the same level of recommendation accuracy in a very different manner (i.e. being correct on non-overlapping sets of users and items). As such, one could argue in favour of focusing on the recommendations that these models generate. That is, we now define model change as differences in model *output*. To avoid biases present in real user histories, we choose to generate k pseudo-user histories using the following procedure. For each pseudo-user, we first generate a uniformly distributed random integer N between N_{min} and N_{max}, which defines the number of items in this user's history. Second, we generate this pseudo-user's history by sampling N items from the dataset without replacement. During sampling each item is given a uniform probability of getting chosen.
This produces a binary matrix $\boldsymbol{X} \in \mathbb{N}^{k \times |I|}$, where $\boldsymbol{X}_{ui} = 1$ when the item i was sampled for the pseudo-user u. For example, when the dataset contains 5 items and we generate $k = 3$ users with $N_{min} = 2$ and $N_{max} = 4$, we might get the following matrix.

$$\boldsymbol{X} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

This matrix $\boldsymbol{X}$ will be used as input to the recommendation models for which we want to evaluate staleness.

Let the updated model at slice s be ϕ_s. For each slice, we compute recommendation scores using an up-to-date model ϕ_s, and an outdated model ϕ_0. Let $P = \phi_s(\boldsymbol{X})$ and $Q = \phi_0(\boldsymbol{X})$. To calculate the change between models, we compute the correlation of the recommendation ranking implied by these two output matrices. We use the Kendall-Tau-b statistic [70] to account for ties in scores. This uses the number of *concordant*, *discordant* and *tied* pairs in the models' outputs. Intuitively, a high Kendall Tau value in a comparison between two model outputs indicates that the models agree on the ranking of most items, and therefore using them in the top-K recommendation scenarios would result in similar outcomes. A low Kendall Tau value indicates that the updated model disagrees with the ranking of the old model. So staleness occurs in ϕ_0, as using ϕ_s in top-N recommendation results in significantly different recommendations.

For each row u we consider all pairs of items (i, j) for which $i < j$ and items i and j have received a non-zero score in either P_r or Q_r.

3.3.2 Estimating Information Gain

The basis of our scheduling method is that we estimate the amount of information gained from events collected since the last update.

Number of events (EV) We assign an equal information value to each event collected since the previous update. So we compute the amount of new information at time T_s since the last update timestamp T_l as

$$\text{information}_{\text{EV}} = \sum_{(u,i,t)\in D:T_l<t\leq T_s} 1 = |\{(u,i,t) \in D | T_l < t \leq T_s\}| \tag{3.1}$$

Inverse predicted relevance (IPR) The assumption that each event contains the same information for the recommender system is often an oversimplification. Unexpected interactions are more valuable than expected ones because they indicate the inability of the model to predict a user's behaviour. To encode the informational value of an event to the model, we propose to use the inverted normalised recommendation score. High recommendation scores will give rise to lower information values and vice versa. Intuitively, if the recommendation model already *expected* the new user-item interaction, it holds less information than when it is *unexpected*. Formally, let $\phi_t : U \to \mathbb{R}^{|I|}$ be the model deployed at time t, and let $\phi_t(u,i) \equiv [\phi_t(u)]_i$ denote the i^{th} output (i.e. the recommendation score for item i). We additionally assume recommendation scores to be normalised to the unit interval. Then, the information value collected at time T_s since the last update timestamp T_l is computed as:

$$\text{information}_{\text{IPR}} = \sum_{(u,i,t)\in D:T_l<t\leq T_s} \frac{1}{\phi_{T_l}(u,i)} \tag{3.2}$$

Output correlation (CORR) Rather than estimate the value of an event, we can also look at the effect the collected events had on a model. If a model computed with the new

events would recommend significantly different items to similar users, we can surmise that a large amount of information has been gathered. Conversely if the output is closely correlated, we can assume that the new interactions did not carry much information. Let M_l be the model computed at the time of the last update (T_l), M_s the model computed at T_s and U' a fixed sample of users. We compute CORR = correlation($M_l(U')$, $M_s(U')$), as described in Section 3.3.1. Ideally we would like to compute the change of the model deployed in production, however this would require updating that model, which is exactly the action we are aiming to avoid doing unless necessary. Instead, we use a more cost-efficient but less accurate proxy-model that can be updated frequently without high cost. In our, experiments we use a "recently popular" model to compute correlations, but more sophisticated, efficiently computable, models can be used in exactly the same way.

3.3.3 Scheduling Model Updates

At each discrete timestamp t, our scheduling method needs to decide whether or not to schedule an update of the model. The scheduling method should most efficiently use an allowed budget to update the model such that performance over the whole period is maximized. For the first two methods EV and IPR, if the amount of collected information since the last update at timestamp T_l is above a threshold δ, an update will be scheduled. For CORR an update is scheduled once the model correlation falls below the threshold δ.

The "*optimal*" value for δ yields the highest model accuracy, within a specified update budget. As such, to find the optimal value for the threshold, our proposed methods should disregard any values that would introduce too frequent updates.

The allowed update frequency f_{update} depends on the available budget for model computations and the computational complexity of the model m used.

$$f_{\text{update}} = \frac{\text{budget}}{\Delta t \cdot \text{cost}(m)}.$$

Where $\frac{\text{budget}}{\Delta t}$ is the available budget for a certain period, and cost(m) the monetary cost of training the model once. Evidently, models that require more resources, can be updated less frequently.

Using EV the value for δ_{EV} is the number of events before scheduling an update. Given a requested number of updates, this can be computed exactly. If f_{EV} is the average frequency of events (events / day) and f_{update} the requested update frequency (updates / day), then δ_{EV} is computed as:

$$\delta_{\text{EV}} = \frac{\text{information}_{EV}}{\text{update}} = \frac{f_{\text{EV}}}{f_{\text{update}}} \tag{3.3}$$

For the two other methods, we cannot compute an exact value, and so offline optimisation on a validation dataset is required. That is, for a range of values for δ, we calculate the number of updates that *would* have been scheduled by IPR or CORR. Then, we pick the maximal value of δ that remains within the prespecified budget.

Dataset	|D|	|U|	|I|	Period
Adressa	2 532 729	228 462	2 790	7d
Globo.com	2 722 355	218 228	9 759	17d
Industry *News*	3 323 941	239 149	3 378	7d
CosmeticsShop	7 877 677	483 080	27 019	152d
Industry *Retail*	12 066 513	995 651	15 712	46d

Table 3.1: Properties for the datasets used in the offline experiments.

3.4 Experimental Results

3.4.1 Datasets

In order to validate the proposed methods in a variety of domains, we use publicly available datasets for both news and retail use-cases, as well as two proprietary industrial datasets. For the news use-case, we use the Adressa [46] and GLOBO [31] datasets and the proprietary "*News*" dataset. For the retail use-case, we use the CosmeticsShop Kaggle dataset, as well as the proprietary "*Retail*" dataset.[2][3]

As is common in the literature – we pruned users and items with very low numbers of interactions from the dataset [14]. In doing so, we significantly decrease the computational complexity of model training without significantly impacting the results obtained. As our experimental setup consists of an iterative model training and evaluation process, this benefits the reproducibility of our work with reasonable computational resources. We require users to have interacted with at least 3 items, and 10/50 interactions per item for the news/retail datasets respectively.

Table 3.1 shows summary statistics of the pre-processed datasets. The news datasets contain fewer items, and are collected over much shorter periods of time compared to the retail datasets. The train-test timestamp T_0 was kept constant per dataset for each experiment. For Adressa and the proprietary News dataset, we used the last two days of the data as test data. For Globo, we used the last five days. For CosmeticsShop we used the final month as test data. For the proprietary Retail dataset, we used the last two weeks. When evaluating algorithmic performance, we used sliding windows of 15 minutes for the news datasets, and 6 hours for the retail settings. This significantly reduced granularity for retail datasets is justified by our findings that models for retail take longer to change compared to news models.

[2]https://www.kaggle.com/mkechinov/ecommerce-events-history-in-cosmetics-shop
[3]https://rees46.com/

Dataset	Algorithm	Runtime (s)	Memory (GB)	Monthly Cost (USD)
Adressa	ItemKNN	35	1	0.47
	EASEr	44	1.2	0.60
	Mult-VAE	2 120	3	695
	BPR-MF	7 538	3	1 180
Globo	ItemKNN	34	3	0.46
	EASEr	302	8	4.1
	Mult-VAE	3 275	4	1 074
	BPR-MF	7 627	3	1 080
CosmeticsShop	ItemKNN	94	11	2.6
	EASEr	3 490	32	330
	Mult-VAE	15 800	32	1 346
	BPR-MF	-	> 60	-

Table 3.2: Computational requirements for public datasets and algorithms. Monthly cost assumes computing the model every hour. If training takes longer than one hour we make the assumption that training is only started once the previous model is done computing. For Mult-VAE we used the default parameters for hidden and bottleneck layer sizes (600 and 200). BPR-MF models were trained with embedding size = 100. Mult-VAE and BPR-MF were both run for 50 epochs. The BPR-MF computation for CosmeticsShop dataset ran out of memory on our testing machine with 60 GB RAM.

3.4.2 Recommendation Algorithms

Table 3.2 lists the recommendation algorithms we considered for comparison in our offline experiments. Although we acknowledge that real-life industrial recommender systems are much more complex than any of these algorithms, we do believe that they are prototypical and can represent such systems for which the same principles of drift and scheduling apply. The cost per update is the estimated cost in USD for training the model once on a dedicated virtual machine (VM) on Google Cloud Services [4]. The approaches that benefit from dedicated GPUs (BPR-MF [107] and Mult-VAE [80]), were given a single NVIDIA Tesla P100. The computation time was computed on a machine with 14 virtual cores and 60 GB of RAM memory. As BPR-MF required more RAM than available on the CosmeticsShop dataset in our test setup, it is left blank in Table 3.2. For deep learning approaches, the introduction of a GPU increases costs dramatically, as well as their relatively higher run times. These costs are based on the training times of these models on the available data. In real world applications, the data available usually spans a longer period, more users and more items than we have available in the offline datasets. Because of this, we can expect practical estimates of the cost per update to be multiple times higher than the values reported here. Nevertheless, our method is unaffected by this scaling factor. By adjusting the cost estimate to the data at hand, we can still schedule updates on a given budget. Parameters used in the experiments were tuned using grid search on the validation set.

Because of the long training times for Mult-VAE and BPR-MF, our experiments that rely on iterative model retraining to simulate online behaviour would require several weeks of computation time. To improve reproducibility of the work whilst remaining relevant for

[4] https://cloud.google.com/products/calculator/

state-of-the-art methods, we therefore focus on two scalable and highly competitive item-based collaborative filtering algorithms: Item-kNN and $EASE^R$. For a short description of the models see Chapter 2.

We released all source code to reproduce the results on the public data presented throughout the following sections.

3.4.3 Model Staleness

In this section we investigate why it is important to update recommendation models. First we show that models grow stale when they are not updated. In a second experiment, we show that (frequent) model updates mitigate model staleness and are necessary to achieve adequate performance.

3.4.3.1 Existence of Model Staleness

We investigate the existence of model staleness by looking at how a models performance evolves as it gets older and how much the output changes between models trained at various points in the timeline. We follow the methodology defined in Section 3.3.1.

In Figure 3.1, we show how the accuracy of a model degrades over time for each of the datasets. To reduce the variance, and impact of time of day in evaluation, we compute the degradation for 12 different starting timestamps, each separated by one hour. As expected, models for news datasets suffer from staleness after only a couple of hours, while models in retail contexts remain similarly performant for a longer time. Interestingly, models on Globo and Adressa behave differently. On Adressa the degradation is much stronger than for Globo. The rate of model staleness is specific to the dataset, and thus it should be taken into account when choosing a training schedule in production.

To investigate the change in model output, we use $N_{\text{min}} = 4$, $N_{\text{max}} = 10$ such that the sampled histories contain between 4 and 10 interactions and $k = 5\,000$ samples for Adresssa, $k = 10\,000$ for Globo, and $k = 20\,000$ for CosmeticsShop.

The correlation results in Figure 3.2 extend the results of our analysis on accuracy. Based on their output, the models for all datasets grow stale. However, this change is not directly related to the chosen accuracy metrics. For example, a week-old ItemKNN model for CosmeticsShop still has an accuracy of 95% compared to the up-to-date model, despite their outputs only having a correlation of 0.7. Therefore, we remark that the model changes faster than it becomes stale, and this change could affect metrics such as diversity or fairness. This highlights that the number of required updates depends strongly on the chosen success metric as well. In this chapter, we limit the scope of our study to accuracy metrics.

Both decreases in accuracy and correlation of output between models confirm for recommendation models that, if we want high performance, we need to make sure the model is up-to-date [23].

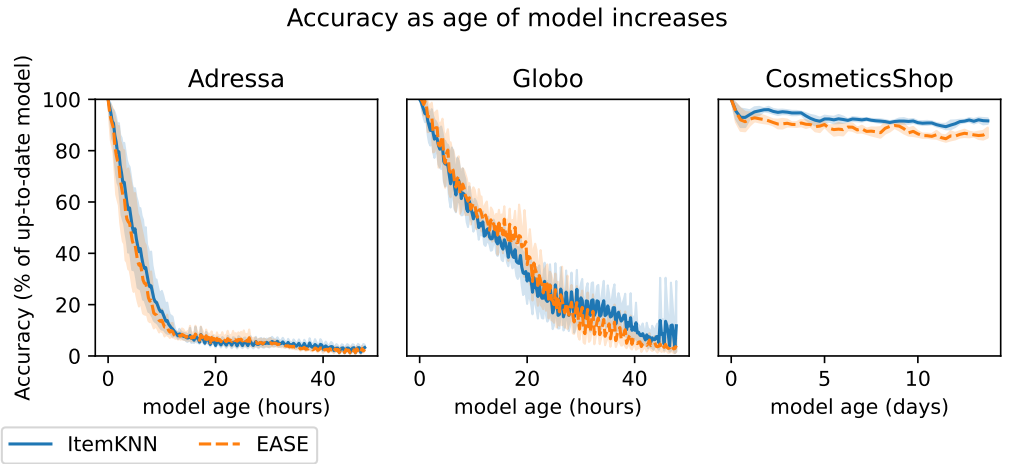

Figure 3.1: The y-axis shows the accuracy of the model as a percentage of the accuracy of the up to date model. Accuracy decreases much faster for news datasets, compared to the mostly stable performance on the retail dataset, both for EASE^R (Dotted line) and ItemKNN (full line) models. The shaded area shows the 90% confidence interval.

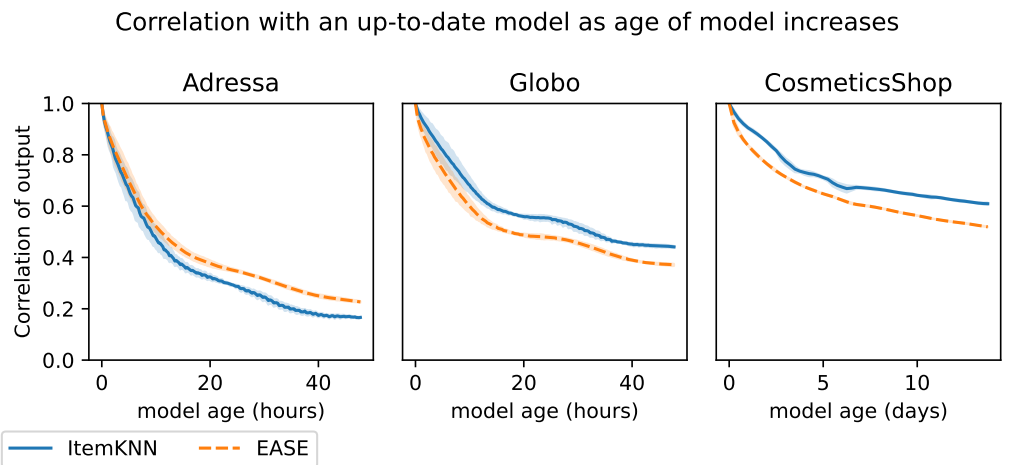

Figure 3.2: The y-axis shows the correlation between the output of an out-of-date model, and the up-to-date model. The x-axis shows the age of the out-of-date model. The shaded area shows the 90% confidence interval.

3.4.3.2 Impact of model staleness on model performance

We investigate how scheduling can mitigate the staleness effect, by evaluating how well a traditional regular interval scheduler mitigates the staleness of models. Specifically, we vary the update frequency to investigate the benefit gained from more frequent updates, as well as the costs associated with them.

Based on the results in Section 3.4.3.1, we expect more frequent updates to result in higher accuracy. Figure 3.3 reflects this expectation. For both news datasets, infrequent updates lead to lower performance than the more frequently updated models. However, we do notice a diminishing return on the number of updates, illustrated by the sections of the plots with zero gradient. For example, updating twice per hour (96 updates) to three times per hour (144 updates) is not significant. Updating from a single update every four hours, to two updates per hour does show a significant increase ($p < 0.05$) in performance.

For the retail dataset, CosmeticsShop, the stable performance of out-of-date models shown in Figure 3.2 suggests that the benefit of increased updates is low. This expectation is confirmed here as well, as further increasing the number of updates past 14 for the period of 14 days yields only minimal accuracy improvement.

For both retail and news datasets, there is a significant performance gap between the ItemKNN results and the $EASE^R$ accuracy scores. While it is not the goal of this chapter to investigate the relative performance of these models, it is interesting to remark that more complex models that have a higher accuracy, also require a higher computational cost. Therefore, it is necessary to consider the estimated cost of each update in addition to the number of times a model is updated.

Given the estimated costs per update in Table 3.2, the second row of plots in Figure 3.3 shows the cost associated with the updates and resulting model performance. Costs scale linearly with the amount of updates, but vary strongly from model to model. For example, computing the $EASE^R$ model for CosmeticsShop once is more expensive than computing the ItemKNN model 35 times. When deploying a recommender system in production, it is necessary to take this cost into account. Neural network approaches are popular in the literature, and, as we have shown, they often require high computation costs to train. In many use-cases, simpler up-to-date models can perform better within tight budget constraints. Additionally, if we can decrease the cost of a single model computation, we can update more frequently and increase accuracy as a consequence. This highlights the importance of incremental models.

Similarly, faster-to-compute approximate approaches can also increase accuracy compared to their exact counterparts when the budget is constrained – despite losing accuracy in static evaluation scenarios. As an example, in their paper on "Markov Random Fields for Collaborative Filtering", Steck [122] reports that the approximate version of the algorithm requires five times less computation time, while only losing 0.8% accuracy. This means that the cost per update for the approximation is at least five times lower than that of the exact model. As such, at constant cost, the approximate model can be trained five times more often. This compensates the theoretical loss in accuracy in many production settings. For the remainder of this chapter, we return to an abstract notion of cost to focus on the impact of well-timed model updates on accuracy.

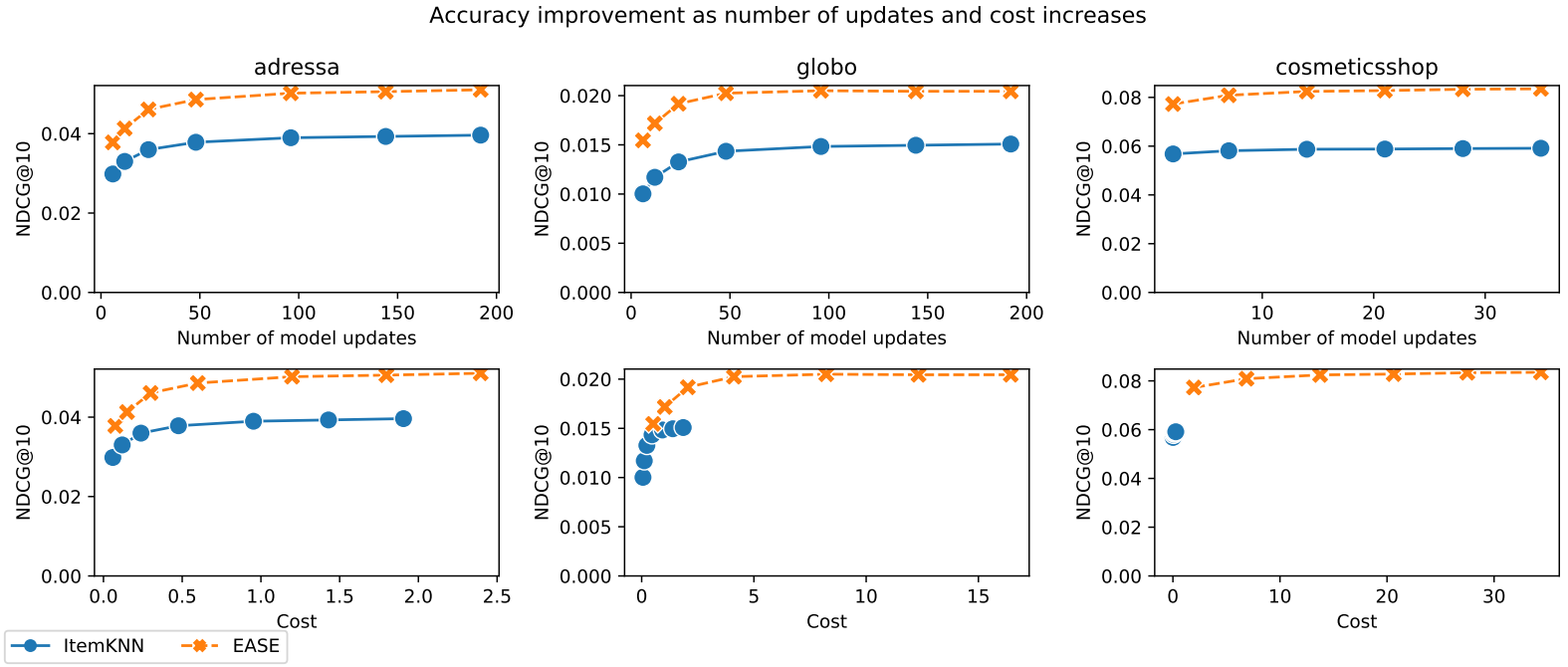

Figure 3.3: The y-axis shows NDCG@10 of the model, the x-axis shows the number of times the model was updated and the cost associated with these updates. Similar results are obtained with Recall@10, which we omit here for brevity. Costs for ItemKNN are far lower, as the model trains much faster, especially when the number of items grows. On cosmeticsshop the cost difference between ItemKNN and EASE is so big, that all ItemKNN point are around the same (low) cost.

3.4.4 Comparing Scheduling Methods

In order to evaluate which of the methods for estimating the information gain described in Section 3.3.2 performs the best, we compare them for Item-kNN and EASE^{R} on all datasets.

For each of our two types of datasets, we use two different update frequencies. For the news datasets (evaluated over a period of 48 hours), we use schedules with 24 and 48 updates (f_{update} = 12/day and 24/day). For the retail dataset (evaluated over 14 days), we use schedules with 7 and 14 updates (f_{update} = 0.5/day and 1/day). We choose these numbers of updates, such that the amount of model updates does not yet counteract staleness entirely, as visualised in Figure 3.3. Thus, leaving room for smarter scheduling to make a measurable difference. Furthermore, these update frequencies are typical in production settings.

In order to make results between schedulers comparable, we ensured that all schedulers updated the model exactly as often as requested. Simulating a situation where the threshold selection on the validation dataset results in exactly the same amount of updates on the test dataset. To do this, we started by computing the threshold as defined in Section 3.3.3, and then slightly increased or decreased it until the right amount of updates are scheduled during the test period.
While this procedure goes against good practices for optimising parameters, we show in Figure 3.4 that given more data than available in the offline datasets, we would be able to accurately estimate the correct threshold value that schedules exactly the right amount of updates for a given frequency. We visualise how accurate our estimates of the thresholds really are, by plotting the amount of updates scheduled by each of the methods, compared to the amount requested. We only present the results for all schedulers for the Adressa dataset, due to run-time constraints. Compared to the equal time schedule, which by design schedules the right amount of updates, the fitted parameters mostly schedule the expected amount of updates. The EV method is the strongest outlier, scheduling fewer updates than expected. This is a direct consequence of the event frequency that changes significantly throughout the dataset, showing that the EV method is more susceptible to variations in amount of traffic than the other two. In order to validate that in practice the right amount of updates would be scheduled, we also show the results on a real-world news dataset. With two months of validation data and 14 days of test data, the results confirm that given more data, the day-to-day variance is removed, such that in realistic scenarios the expected amount of updates will be scheduled during the test period, and our proposed methods compute the right thresholds. It is therefore an effect of the size of the offline datasets, that we need to modify our thresholds after computing them on the validation dataset to obtain a fair comparison.

The results of experiments comparing scheduling methods are presented in Tables 3.3 and 3.4.

Key empirical observations from these experiments are as follows:

For the large majority of settings, at least one of the proposed novel methods outperforms the widely used “equal time” baseline. This confirms that smart scheduling regimes can improve recommendation performance at constant cost.

The IPR method generalises the best to all datasets, either performing superior or close

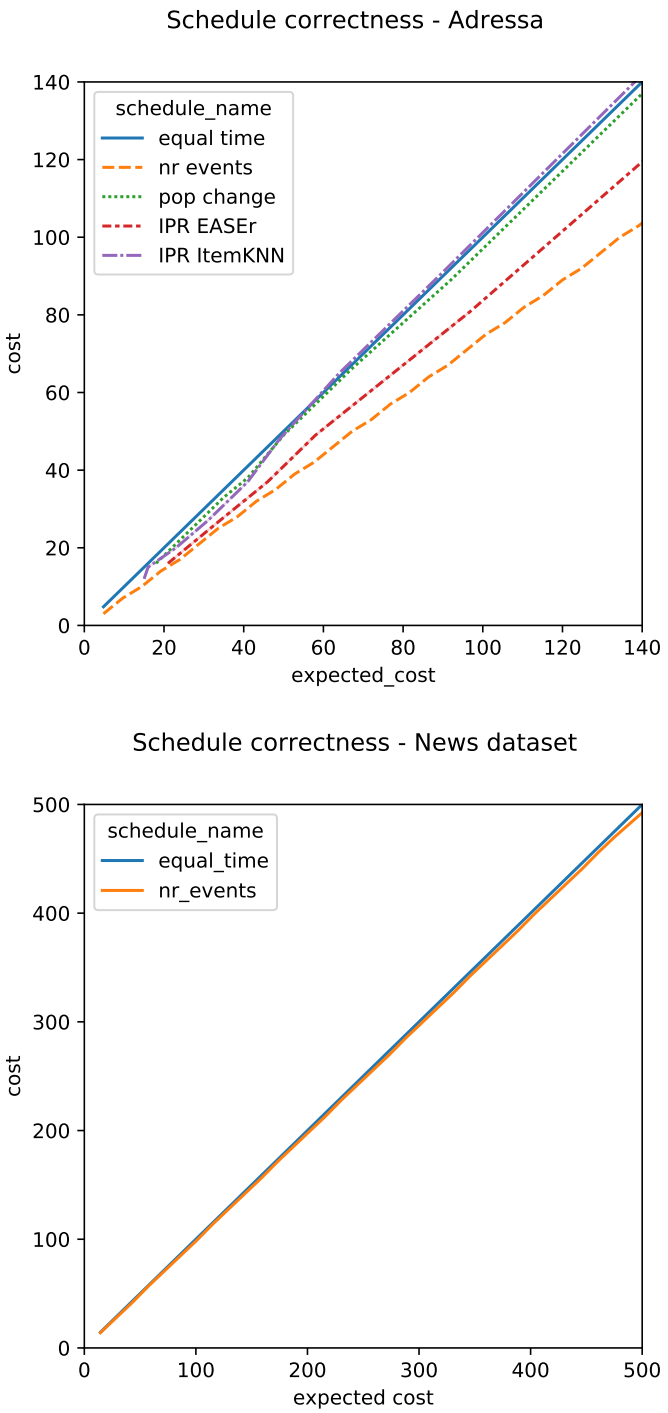

Figure 3.4: The x-axis shows the number of updates requested during optimisation, the y-axis shows the number of updates effectively scheduled during testing. The top plot shows results for the Adressa dataset, using 2 days of validation data to select the threshold parameters. The bottom plot presents the schedule correctness on the NEWS dataset for the number of events scheduling threshold. We perform this only for this schedule, as it is the one with the highest variance in the experiments on the small dataset.

	ItemKNN			
	NDCG@10	**Recall@10**	**NDCG@10**	**Recall@10**
Number of updates	24		48	
Adressa				
equal time	3.60	6.82	3.78	7.21
EV	**3.67**	6.98	**3.83**	7.30
CORR	3.53	6.74	3.76	7.26
IPR	3.64	**7.00**	3.82	**7.38**
Globo				
equal time	1.33	2.14	1.43	2.33
EV	1.36	2.21	1.44	2.35
CORR	1.36	2.21	1.44	2.35
IPR	**1.38**	**2.23**	**1.45**	**2.36**
Industry news dataset				
equal time	3.79	7.00	4.04	7.49
EV	**3.91**	**7.25**	**4.11**	**7.63**
CORR	3.77	6.98	3.98	7.35
IPR	3.88	7.20	4.10	7.61
Number of updates	7		14	
CosmeticsShop				
equal time	5.81	8.13	**5.87**	8.21
EV	5.79	8.12	5.84	8.17
CORR	**5.82**	**9.18**	5.85	9.23
IPR	5.79	9.13	5.86	**9.24**
Industry retail dataset				
equal time	12.17	15.63	12.22	15.69
EV	12.16	15.62	**12.23**	**15.70**
CORR	12.10	15.53	12.22	15.68
IPR	**12.18**	**15.65**	**12.23**	15.69

Table 3.3: The NDCG@10 and Recall@10 results in % with the ItemKNN algorithm for all considered datasets. Experiments on news datasets were evaluated over a period of 48 hours, on retail datasets over a period of 14 days. The results for the best scheduling methods are shown in bold per configuration.

	$EASE^R$			
	NDCG	**Recall**	**NDCG**	**Recall**
Number of updates	24		48	
Adressa				
equal time	4.61	8.58	4.86	9.06
EV	4.69	8.77	4.92	9.20
CORR	4.65	8.78	4.91	9.29
IPR	**4.73**	**8.93**	**4.96**	**9.40**
Globo				
equal time	1.92	3.18	**2.03**	**3.38**
EV	1.92	3.20	2.02	3.37
CORR	**1.97**	3.26	2.02	3.36
IPR	**1.97**	**3.28**	2.02	**3.38**
Industry news dataset				
equal time	4.61	8.08	4.94	8.75
EV	**4.77**	**8.41**	**5.04**	**8.94**
CORR	4.55	7.97	4.83	8.51
IPR	4.73	8.32	5.01	8.89
Number of updates	7		14	
CosmeticsShop				
equal time	8.09	10.99	8.24	11.21
EV	8.03	10.94	8.20	11.15
CORR	**8.12**	**12.32**	**8.25**	12.53
IPR	8.10	12.31	**8.25**	**12.55**
Industry retail dataset				
equal time	**14.79**	18.49	14.86	18.61
EV	14.78	18.49	**14.87**	18.62
CORR	14.71	18.38	14.86	18.61
IPR	**14.79**	**18.50**	**14.87**	**18.63**

Table 3.4: The NDCG@10 and Recall@10 results in % for $EASE^R$ algorithm on all considered datasets. Experiments on news datasets were evaluated over a period of 48 hours, on retail datasets over a period of 14 days. The results for the best scheduling methods are shown in bold per configuration.

second. This confirms that the method is able to detect the right signals to decide on updating the model, regardless of the context in which it is used.

Since both IPR and CORR improve over EV in some settings, we confirm our hypothesis that not all events carry the same amount of information. Depending on the context, it is necessary to take this into account to find the optimal schedule.

Each of the three methods is model-agnostic, and can be used with any model that generates recommendation scores. The EV and CORR methods do not depend on the deployed model for choosing their updates. IPR conversely does depend on the relevance scores predicted by the recommendation model, but the results for ItemKNN and EASE show that it generalises to different models.

Improvements with novel methods are most notable on the Recall@10 metric, rather than on the NDCG@10 metric. This can be explained by the fact that new items typically get recommended at the bottom of the top-K. Recall is much more sensitive to these lower ranked, but correct new items, compared to NDCG, which focuses on ranking the correct items at the top of the list.

3.4.5 Online Experiments

In order to validate the offline results in this article, we also performed several online experiments on both news and retail websites.

For the news use-case we got the opportunity to use a Dutch-language, local, newspaper website. The staleness rates in the offline experiments showed patterns matching the Adressa dataset, suggesting that models will grow stale quickly when not retrained.

For the retail use-case we used two very different online webshops. A "traditional" retail webshop, showing similar staleness patterns as CosmeticsShop and a "flash" retail webshop, where items are only available for a limited time, whose offline staleness patterns showed behaviour between the news and retail datasets. Models do grow stale, but not as quickly as they do for news.

Model staleness To confirm that the results for our model staleness hypotheses presented in Section 3.4.3.2 also hold in an online setting, we performed three randomised controlled trials, colloquially known as an A/B-test, with control/treatment corresponding to the same recommendation model updated at different intervals. In these trials, we do not yet make use of the new scheduling methods we proposed, but instead use the traditional fixed schedule.

In the first of these trials we verified that model staleness impacts performance quickly in a fast moving news context, such that faster updates result in better performance. For the control group, we updated the model every 30 minutes and the treatment group had its model updated at an increased rate of once every 15 minutes. Based on our offline experiments, we expected the treatment group to outperform the control group due to the reduced staleness in the serving models. This was confirmed after a 2-week trial, in which the treatment group showed a 2% relative increase in click-through-rate (CTR) compared to the control group.

In a second trial on the traditional webshop, we verified that fewer updates did not reduce recommendation performance. For the control group in this trial the model is updated every 3 hours, while for the treatment group it is updated once per day. We expected no significant difference in CTR between the two groups, given the offline results. After a trial that ran for 4 weeks with more than 1.5 million recommendation opportunities for each of the groups, indeed no significant difference was found between the two schedules. This is an important insight: indeed, reducing the number of model updates by a factor of 8 implies reducing the computational costs for this model by a factor of 8.

The final trial was held on the flash-retail website. Our expectation was that increasing the amount of model updates from 4 per day (control group) to 8 per day (treatment group) would prove beneficial on both CTR and conversion. This was confirmed after the 10-day trial, with the treatment group showing a significant improvement in both CTR (+20%) and conversion (+25%) over the control group.

Smart scheduler To extend our evaluation of the smart scheduling methods proposed, we performed two trials where a smart scheduling approach was compared to a baseline fixed schedule approach. During these trials we opted to use the scheduler based on the CORR information metric, computed on a "recently popular" model, for practical reasons. While the IPR scheduler performed better in many of the offline settings, it was much harder to integrate into the recommendation pipeline.

For the first trial, on the news website, we used the winning treatment from our previous test (identifying model staleness) as the control group in this test. The models are updated according to a fixed schedule, with an update every 15 minutes. The treatment group used the CORR scheduler, whose threshold was fitted to schedule 48 updates per day using a validation dataset. Note that the treatment will perform half as many updates as the control group, which without the scheduler was significantly worse in the previous test. This trial was carried out on 3 different lists of recommendations on the website, each shown on different pages. After a trial of one week, no significant difference was found between the two groups for two out of three lists. On one of the lists, the control group performed significantly better than the treatment group, though the improvement was small ($< 1\%$) especially given that it required twice the cost. As such, this test confirms that the CORR scheduler manages to schedule its updates at better timepoints, so it requires only half of the updates—and half of the budget— to get the improved performance we previously only found using 96 updates per day.

The second trial was held on the flash retail website. During this trial the control group received recommendations from a model updated 8 times per day, and the models for the treatment group were scheduled using the CORR scheduler whose threshold was fitted to also schedule 8 models per day. Over the evaluation period of 2 weeks, the CORR scheduler method showed a significant ($p < 0.05$) improvement of 2% in CTR (relative) over the control group.

Given the positive online results of the CORR scheduler, and the better offline performance of IPR, we are considering it future work to also implement the IPR scheduler online, and verify its offline superiority over the popularity change scheduler in online trials.

3.5 Conclusions

In this chapter we have demonstrated the effect of retraining and model staleness in different environments and for a variety of collaborative filtering approaches, and we have studied the cost-accuracy trade-off. Achieving higher accuracy requires increased computational costs up to some asymptotic point, after which more updates have no further benefit. In real world applications of recommendation systems we cannot ignore this cost of computing models and therefore it is paramount to find an optimal balance between cost and accuracy. Our results indicate the importance of models that can be efficiently and incrementally trained for real-world applications – because they allow frequent updates while keeping costs low.

We have proposed a generic method to create a smart schedule for retraining or updating models, which results in higher accuracy than retraining at fixed intervals given the same resources. Our scheduling approach uses a heuristic to comply to the budget constraint. Applying this budget constraint to control methods and drift detection methods is an interesting avenue to further improve the use of resources in practice.

In our experiments, we hinted at the fact that up-to-date models also have an impact on other metrics, such as fairness or diversity. It remains interesting for future work to investigate this impact thoroughly.

Chapter 4

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

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]

[1]This chapter is based on "Are we Forgetting Something? Correctly Evaluate a Recommender System With an Optimal Training Window." Robin Verachtert, Lien Michiels and Bart Goethals. In Proceedings of the Perspectives on the Evaluation of Recommender Systems Workshop 2022.

4.1 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 through temporal or leave-last-one-out splits [13, 61, 67, 84]. 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 [22]. 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 [64]. But is this also true for more complex, personalised recommendation algorithms?

In this chapter, 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 chapter, we will refer to the maximal age of an event used in training as the hyperparameter δ_{in}.

We investigate and answer the following three questions:

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

Additionally, through our experiments we show that the optimal δ_{in} 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_{in} 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_{in} to future work.

In Section 4.2 we highlight relevant related work. Section 4.3 describes how δ_{in} should be considered a hyperparameter, and how to set up an evaluation to mimic a real-world

scenario. Also in Section 4.3 we present the chosen algorithms, datasets and evaluation metrics. Finally, Section 4.4 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_{in}.

4.2 Related Work

Research in data science has recognised that data drift is an import factor in training high quality models for several decades [20, 40, 72, 133]. More specifically, Fan [36] 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_{in}, is a straightforward way to avoid training poorly performing models [36].

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 [24, 98], a sequential last item prediction split [58] and repeated time-aware splitting [61, 114]. 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. [24], Ludewig and Jannach [85], Quadrana et al. [104] and Bogina et al. [22].

Relevant to our work, Vinagre and Jorge [131] summarised two generic methods for dealing with concept drift in a data-stream. The first is to utilise a predetermined δ_{in} 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 [87] used a contextual popularity algorithm, with δ_{in} equal to five minutes, thirty minutes and one hour, to great success in the CLEF Initiative in 2017. Similarly Ji et al. [64] showed that computing popularity using a small δ_{in} or using fading factors provided a much stronger baseline. Jannach and Ludewig [57] and Jannach et al. [58] 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 [6, 76, 81, 83, 126, 135], similar to Vinagre and Jorge [131]. 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 [42], Sequential Rules [57], VS-KNN [57], and GRU4Rec [53]. In the wake of GRU4Rec, more and more deep learning approaches have been proposed that incorporate sequential and/or temporal information. [e.g. 82, 84, 124].

Recent reproducibility studies have challenged the performance of these complex deep learning methods in a variety of domains. In two recent works, Ferrari Dacrema et al.

[37, 38] found that "11 of the 12 reproducible neural approaches can be outperformed by conceptually simple methods", such as ItemKNN or UserKNN. Ludewig et al. [86] 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.

4.3 Methodology

4.3.1 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 [13, 67, 84], or iterative revealing [86].

As highlighted in Chapter 2, Section 2.4, this scenario has leakage issues [65]. Instead we follow the `TimedLastItem` scenario, which tackles these leakage issues, while maintaining the sequential evaluation. 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 [114] for future work.

4.3.2 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_{in} values as they experience weaker data drift, while performance on news datasets suffers more drastically when δ_{in} 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 [46] 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 [17], the CosmeticsShop Kaggle dataset [69] 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

Dataset	$\|\mathcal{D}\|$	|U|	|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
Cosmetics	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

Table 4.1: Properties for the datasets used in the offline experiments

7-day windows. 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 4.1. 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 [29]. 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.

4.3.3 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_{in} for each of them.

- Popularity
- Item-kNN [32, 113]
- IKNN Ding (Item-kNN variant with time-decay) [33]
- IKNN Liu (Item-kNN variant with time-decay) [81]
- EASE^{R} [121]
- GRU4Rec [134]
- SR (Sequential Rules) [86]

For a short description of these algorithms, see Chapter 2, Section 2.5. An in-depth discussion of the Item-kNN variants with time-decay, can also be found in Chapter 5.

4.3.4 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) [115], Catalog Coverage (Coverage) [102], Recall [115] and Mean Reciprocal Rank (MRR) [32] 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.

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 [43].

4.3.5 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_{in} to be inspected over a large range of potential values.

Rather than using a random search we utilised the Tree-structure Parzen Estimator [18] as implemented in the Python `hyperopt` library[2] [19]. 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.

[2] https://hyperopt.github.io/hyperopt/

4.4 Results

dataset	RETAIL	Yoochoose	Cosmetics	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

Table 4.2: Optimal δ_{in} values found during optimisation, rounded to the nearest hour.

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].

4.4.1 RQ1: "How does the optimisation of δ_{in} impact the individual performance of an algorithm?"

In Table 4.2 we present the optimal values for delta found during optimisation, and in Table 4.3 we present the corresponding NDCG@10 values. We compute an NDCG value for both the model trained on all training data ($\delta_{in} = \infty$) and on the optimised δ_{in} ($\delta_{in} =$ optim).

The optimal choice of δ_{in} 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

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

dataset	RETAIL		Yoochoose		Cosmetics		NEWS		Adressa	
delta	∞	optim	∞	optim	∞	optim	∞	optim	∞	optim
EASE^R	-	-	-	-	4.8	4.6	2.0	5.5	0.8	7.0
GRU4Rec	-	-	13.6	13.6	3.3	2.9	3.7	3.2	4.0	3.9
ItemKNN	6.4	6.4	16.5	17.8	4.9	4.9	1.3	4.9	0.4	5.4
Popularity	0.7	0.8	0.4	1.1	0.9	1.1	1.0	4.8	0.4	12.6
SR	9.3	9.3	19.0	20.7	7.2	7.2	3.2	4.5	3.6	4.5
IKNN Ding	8.5	8.5	17.1	18.5	6.4	6.4	1.5	5.8	0.6	6.4
IKNN Liu	8.8	8.8	18.8	18.7	6.4	6.4	2.6	3.6	3.9	3.9
correlation	1.00		1.00		1.00		-0.43		-0.71	

Table 4.3: NDCG@10 in % for optimised δ_{in} values and $\delta_{in} = \infty$. At the bottom of the table we report the correlation between the ranking of algorithms trained with $\delta_{in} = \infty$ and the one with optimised δ_{in}.

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 4.1 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_{in} 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_{in} values. CosmeticsShop is a very stable dataset, and most algorithms perform optimally using almost all of the data (The maximal value for δ_{in} is $124 \cdot 24 = 2976$ hours). For RETAIL, we note that the optimal δ_{in} is usually smaller than on CosmeticsShop, but the performance gains are minimal. This implies 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_{in} 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_{in} 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.

4.4.2 RQ2: "Does the optimisation of δ_{in} change the relative performance between the algorithms?"

We compare how the rankings of algorithms sorted by NDCG change if we go from $\delta_{in} = \infty$ to an optimised δ_{in}. For this comparison we use Kendall's Tau correlation between the two rankings of the algorithms [70]. We report these correlations at the bottom of Table 4.3. 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 $\delta_{in} = \infty$, the time- and sequence-aware approaches show superior performance, however, this is no longer the case given an optimal δ_{in}. The baseline methods surpass the deep learning methods and now perform the best.

dataset	**RETAIL**		**Yoochoose**		**Cosmetics**		**NEWS**		**Adressa**	
delta	∞	**optim**	∞	**optim**	∞	**optim**	∞	**optim**	∞	**optim**
$EASE^R$	-	-	-	-	60.9	56.8	34.1	24.8	23.2	13.9
GRU4Rec	-	-	71.5	52.8	70.0	66.8	41.0	18.5	34.5	32.7
ItemKNN	94.0	89.9	76.5	63.1	60.0	61.3	25.8	21.2	10.4	16.7
Popularity	0.2	0.1	0.1	0.1	0.2	0.2	3.7	1.5	1.9	0.9
SR	89.7	89.5	85.8	65.4	92.4	92.4	47.8	24.4	41.3	23.2
IKNN Ding	90.9	81.1	88.4	71.9	93.7	94.0	14.3	22.1	14.6	17.0
IKNN Liu	88.2	88.2	78.3	73.2	93.2	94.0	65.7	30.4	71.3	68.6

Table 4.4: Coverage@10 in % for optimised δ_{in} and using $\delta_{in} = \infty$. Reducing δ_{in} usually results in a lower coverage, as older items are no longer recommended.

For the retail datasets we don't see this effect, either $\delta_{in} = \infty$ 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 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_{in}, when it was outperformed on the $\delta_{in} = \infty$ setting.

In most scientific articles, the results would be compared using a $\delta_{in} = \infty$ 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_{in}. If we do not, we risk making the wrong conclusions.

4.4.3 RQ3: How does the choice of δ_{in} impact secondary metrics such as run time and coverage?

In Table 4.4 we present the Coverage@10 results for the algorithm-dataset pairs. We see that in general coverage is lower for the optimal δ_{in}. 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_{in} 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_{in} 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

dataset	RETAIL		Yoochoose		Cosmetics		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

Table 4.5: Runtimes (in seconds) for optimal and non optimised δ_{in}. Runtime is the sum of training and prediction time. Decreasing δ_{in} also decreases the runtime, as less data needs to be processed.

contribute to problematic situations in production settings. In Table 4.5, the run time for the optimisation trials with optimal δ_{in} and maximal δ_{in} 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, there is a small increase in performance when changing to an optimal δ_{in} 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.

4.4.4 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_{in}, the training window was chosen manually by engineers with some input from editors. By performing the optimisation of δ_{in} as suggested in this chapter, we found that the original values were not optimal, and could be improved by using smaller δ_{in} 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 $\delta_{in} = 3h$ was used as our control treatment. During the offline experiments, we found that $\delta_{in} = 1h$ 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 $\delta_{in} = 1h$ 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 $\delta_{in} = 2h$ 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 $\delta_{in} = 6h$, and a second with $\delta_{in} = 10h$. 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_{in} 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.

4.5 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 chapter, 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_{in}. We believe that we have clearly shown that the choice of the δ_{in} matters, both to find the optimal performance of individual algorithms and to make a fair comparison between algorithms. Optimising δ_{in} for each algorithm should be standard practice in the evaluation of recommender systems. Not optimising δ_{in} will favour only the algorithms that account for drift.

Chapter 5

A Unified Framework for Time-Aware Item-Based Neighbourhood Recommendation Methods

For recommender systems to thrive, they need to cope with dynamic environments. A prominent challenge these systems face is that historical interactions can lose relevance. User interests change; what they liked last year might no longer be interesting to them. The relevance of items to the user base also changes, a news article published last week is likely no longer relevant to the majority of the readers. Time-aware recommender systems (TARS) are developed to utilise timestamps to generate better recommendations. Recently, time-aware user-based neighbourhood (UserKNN) models have shown competitive performance in comparative studies, outperforming various deep learning techniques. Unfortunately, UserKNN methods suffer from significant challenges in production. Finding similar users is an expensive operation, and storing a full user-user similarity matrix is usually not feasible. Item-based neighbourhood methods (ItemKNN) are more efficient at predicting items and require less storage to store the model. Time-aware ItemKNN methods exist; however, we find that they are usually not considered in recent studies. In this chapter, we provide a unified framework for time-aware ItemKNN methods, allowing us to identify and fill gaps in the literature. Further, we show that these models are competitive on recent large-scale datasets. Finally, we make our code public, so researchers can easily reuse the algorithms in their own baseline comparisons.

5.1 Introduction

Recommendation algorithms are used in highly dynamic environments, they need to cope not just with new items and new users, but also with items that lose relevance, or users developing new interests. Fortunately, we can provide these systems with timestamps for each interaction, thus allowing them to model which items are frequently considered together [51, 135], which items are considered in a similar temporal context [74] or which items usually follow after other items [57, 86].

In this endeavour, deep learning techniques have become the go-to solution [53, 78, 82, 137]. However, recent studies show that user- or session-based neighbourhood models, which are comparatively simpler, perform on par with or outperform more complicated deep learning techniques [38, 57, 75, 86].

Unfortunately, these user- or session-based models are often challenging to use in a real-time recommendation system. Precomputing each user's neighbourhood results in a model that is too large in environments with hundreds of thousands of users. Efficient methods exist to search for a user's neighbours in real-time, through sampling [71], or approximate nearest neighbour search [48]. However, for real-time recommendations, response deadlines are usually so strict (<50ms) that even these more efficient methods struggle to be fast enough.

Conversely, ItemKNN methods build a smaller model, as usually, the number of items is far lower than the number of users. Prediction requires only a single vector-matrix multiplication between the user's history, and the stored similarity matrix, which can be done near instantaneously. This makes them more usable in real-world production scenarios.

In this chapter, we revisit the literature that describes time-aware ItemKNN methods, to show their merit and competitiveness. These methods use timestamp information from interactions to build a better model and to put more weight on recent user behaviour during prediction. Even though these methods were proposed almost a decade ago, we find that they show promising and competitive performance, making them interesting approaches for use in production. We re-contextualise these historic methods by providing a generalised framework, that allows the identification of gaps in the literature. We test the performance of both the historic methods and the unified framework on newer large-scale datasets and demonstrate that they show competitive performance in dynamic scenarios.

This chapter is structured as follows: In Section 5.2 we highlight relevant works in context-aware, time-aware, and sequence-aware recommendation, and introduce the various time-aware ItemKNN methods. In Section 5.3 we describe the generalisation of the time-aware ItemKNN paradigm. Finally, in Section 5.4 we discuss the experimental setup and results.

5.2 Related Work

Almost all online recommendation applications store the timestamp when a user interacts with an item. The availability of this additional data allows algorithms to generate better recommendations.

Context-aware methods use these timestamps to add context to both historical interactions and recommendation targets, for example, considering the time of day [30, 141], the day of the week [119] or seasons [52, 55] as context. For a full overview of context-aware models using time as extra information, we refer the reader to the comprehensive survey by Kulkarni and Rodd [74].

Another common approach focusses on the sequential information the timestamps imply, i.e. one event happens after another [57, 86].

Time-aware algorithms use timestamps to account for the age of interaction or the time between co-occurrences [22, 24]. A final way to use timestamps is to limit training data, thus improving the quality of trained models (even if these do not take into account timestamps or order themselves) [66, 127].

Multiple comparative studies of time- and sequence-aware models found that simple methods such as Sequential Rules [86], Popularity [66], or Session-KNN [86] were strong baselines, outperforming deep-learning methods. In Ludewig et al. [86] VS-KNN, a session-based Nearest-Neighbour method, applying a decay to give more weight to recent interactions when comparing two users, outperformed most other methods. This idea is expanded in STAN, a Sequence- and Time-Aware Neighbourhood method [42]. Anelli et al. [6] also used a weighted userKNN method to achieve competitive results.

These recent comparative studies [85, 86] did not explore time-aware item-based nearest neighbour approaches. However, the success of the userKNN variants leads us to believe they too may achieve competitive results.

ItemKNN methods are despite their age still very popular, thanks to their competitiveness [85, 92], and computational efficiency [62]. Besides being more suitable for online real-time recommendation than user-based methods, itemKNN methods also have an advantage over many other methods such as embedding or factorization methods, in that any new user interactions can immediately be used to give more accurate recommendations.

To identify the existing related work on time-aware ItemKNN models, we started from three surveys written by Campos et al. [24], Bogina et al. [22] and Vinagre and Jorge [131]. For any work they cite that uses time-aware ItemKNN models, we also look in that work's citations. We continue this waterfall search, ending when all relevant mentioned papers were already encountered.

		Ding 2005 [33]	Lee 2008 [76]	Liu 2010 [81]	Liu 2012 [83]	Vaz 2013 [126]	Xia 2010 [135]	Hermann 2010 [51]
Decay Function	Linear						✓	
	Exponential	✓		✓		✓	✓	
	Log				✓			
	Concave						✓	
Decay Input	Age of events	✓	✓	✓	✓		✓	
	Time between events						✓	✓
Decay applied during	training		✓	✓	✓	✓	✓	✓
	prediction	✓		✓	✓	✓		
Similarity Function	Cosine	✓	✓	✓	✓			
	Pearson		✓			✓		
	Conditional Prob.	✓						
	Weighted Cooc.						✓	✓

Table 5.1: Features discussed in time-aware ItemKNN works

We identified the works presented in Table 5.1 that incorporate temporal dynamics in item-based nearest neighbour methods. In these works, we identify two classes of time-aware ItemKNN models.
The first class uses algorithms that apply a decay factor on the interaction matrix, as summarised by Vinagre and Jorge [131] (time-decay). These methods follow the heuristic that newer data should be considered more relevant in building models and/or when predicting the next items of interest [33, 76, 81, 83, 126].
The second class of algorithms looks at the distance in time between visits as additional information (cooc-decay). When two items are visited further apart, they assume that there is a weaker signal of relation than when they are visited closely together [51, 135].

We propose a unified framework for the time-decayed models and show that it is a competitive algorithm, by comparing with strong baselines put forward in Ludewig and Jannach [85] on large-scale datasets.

We find in our experiments that none of the cooc-decay methods is competitive with the other methods tested, while the computation is also far less efficient, losing us the advantage over the user-based approaches. Therefore, we focus on the time-decay methods, ignoring cooc-decay methods, when constructing the framework in this paper.

5.3 Methodology

We generalise the features present in Table 5.1 into a single framework that covers all of their features except the option to decay time between interactions, analogous to Campos et al. [24]. Given a set of users U and a set of items I, the dataset is represented as $\mathcal{D} = \{(u, i, t) : u \in U, i \in I, t \in \mathbb{N}\}$, a set of user-item-time triples. X is the matrix with the last interaction timestamps: $X_{ui} = \max_{u,i,t \in \mathcal{D}_{ui}} t$ if the user u has interacted with the item i otherwise it is 0. We define now as $\max(X) + 1$ to avoid 0 age, then A contains the time since the user last interacted with that item, $A_{ui} = \text{now} - X_{ui}$ for each user-item interaction in $\mathcal{D}$.

Let $s : \mathbb{R}^{|U|} \times \mathbb{R}^{|U|} \to \mathbb{R}$ be the similarity function between two item vectors, $\Gamma : \mathbb{N}^{|U|\times|I|} \to \mathbb{R}^{|U|\times|I|}$ a decay function, and $W = \Gamma(A)$ the weighted interaction matrix. The similarity matrix S is computed such that the pairwise similarity between i and j is $S(i, j) = s(W_i, W_j)$, where W_i and W_j are both columns of the W matrix.

Prediction for a user can be seen as the multiplication of the weighted user history vector, with the computed similarity matrix $\text{pred}(u) = \Gamma'(A_u)S$. Where Γ' is a second decay function, that does not have to be the same as the one used during training.

The methods presented in the literature differ in their choices of Γ, Γ', and s, we next discuss the available choices for these components.

5.3.1 Decay Functions

We identified four different decay functions used in the various time-aware algorithm works. Although it is possible to define infinitely many decay functions, we believe this selection covers all methods proposed in previous work.

- Exponential decay[1]: $e^{-\alpha x}$ with $\alpha \in \mathbb{R}_0^+$
- Linear decay: $\max(0, 1 - \alpha \frac{x}{\max(A_u)})$ with $\alpha \in [0, \inf[$
- Log decay: $\log_\alpha((\alpha - 1)(1 - \frac{x}{\max(A_u)}) + 1) + 1$ with $\alpha \in \mathbb{N}^+ \setminus \{1\}$
- Concave decay: $1 - \alpha^{1 - \frac{x}{\max(A_u)}}$ with $\alpha \in [0, 1[$

Where $x, y \in A_u$ are ages of interactions, and A_u is the row with all the ages of the interactions for a user u.

To help intuition with the differences between decay functions, we plot them in Figure 5.1. Exponential decay is the steepest of the decay functions, after every $1/\alpha$ seconds the contribution of an event will be divided by e. Concave and logarithmic decay both result in similar decay functions. For both, the decay starts slowly, and only the oldest events are strongly decayed. Intuitively, this could benefit use-cases where events maintain

[1]Other formulations for exponential decay exist in the literature, such as α^x [135], or $\alpha^{\lambda \frac{x}{\mu}}$ [125] these can be changed into the formula used, by choosing the hyper-parameters α, λ and μ appropriately

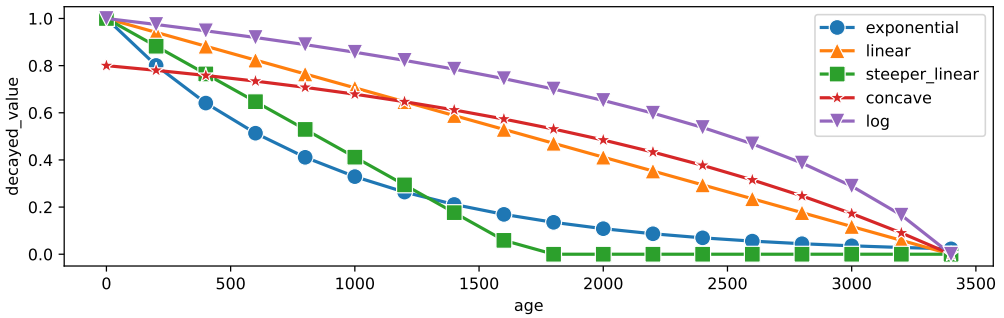

Figure 5.1: Example decay functions.

relevance for building a model, until at some point they become irrelevant quickly. Depending on the value of α, linear decay can be steep or shallow. In the original paper using linear decay [135], the authors specified a maximal value for α of 1. However, in our model, we allow $\alpha > 1$, labelled as the `linear_steeper` function in Figure 5.1. When the decayed value goes below 0, the value is set to 0, and the interaction is effectively forgotten.

5.3.2 Similarity Functions

A similarity function computes the similarity between two items based on the decayed ages of interactions with these items. $s : \mathbb{R}^{|U|} \times \mathbb{R}^{|U|} \rightarrow \mathbb{R}$. We use the following similarity functions extracted from the related work.

Cosine Similarity computes the similarity as the cosine of the angle between the two weighted vectors.

$$\text{Sim}(i,j) = \frac{\sum_{u \in U} W_{u,i} W_{u,j}}{\sqrt{\sum_{u \in U} W_{u,i}^2} \sqrt{\sum_{u \in U} W_{u,j}^2}}$$

Pearson Correlation Accounts for biases in per item scores, assuming that scores dissimilar from the average value for an item are more informative. It is usually used in rating prediction tasks. Intuitively, we feel that this measure might not be well suited to the temporal recommendation use-case, because it removes the decay effect and disregards the assumption that items visited at a similar time are similar. Pearson Correlation is computed as:

$$\text{Sim}(i,j) = \frac{\sum_{u \in U} (W_{u,i} - \overline{W_i})(W_{u,j} - \overline{W_j})}{\sqrt{\sum_{u \in U} (W_{u,i} - \overline{W_i})^2} \sqrt{\sum_{u \in U} (W_{u,j} - \overline{W_j})^2}}$$

Where $\overline{W_i}$ is the average of all, non-zero, decayed ages of interactions with item i.

Dataset	$\lvert\mathcal{D}\rvert$	$\lvert\mathbf{U}\rvert$	$\lvert\mathbf{I}\rvert$	Period
Adressa [46]	2 532 729	228 462	2 790	7d
CosmeticsShop [69]	7 877 677	483 080	27 019	152d
Yoochoose [17]	16 044 427	1 882 684	44 415	182d
Amazon Games [93]	385 543	42 480	14 817	6 928d
Amazon Toys & Games [93]	1 531 360	172 606	69 716	6 939d

Table 5.2: Properties of datasets used in the experiments.

Conditional Probability (Inspired) Similarity Similarity is computed as:

$$\mathrm{Sim}(i,j) = \frac{\sum_{u \in U} W_{u,i} W_{u,j}}{|X_i|}$$

This formula follows the approach taken by Deshpande and Karypis [32] to compute the similarity between items in a normalised interaction matrix. Although this does not result in an actual probability, their and our results show that the formula has value on non-binary matrices. This similarity will favour items that have recently been visited frequently. We, therefore, expect it to perform well in datasets where the trendiness of an item is an important factor.

5.4 Experiments

We largely follow the experimental design presented in Verachtert et al. [127]. Their publicly available code[2] formed the basis for our experiments. For clarity, we repeat the most important concepts of the setup here.

5.4.1 Datasets

We use five large-scale public datasets, namely the news dataset Adressa [46] and the retail data sets Cosmeticsshop [69], Yoochoose [17], Amazon Games and Amazon Toys and Games [93].

We present the properties of these datasets in Table 5.2. Reporting the number of interactions ($|\mathcal{D}|$), number of users ($|U|$), number of items ($|I|$) and the amount of time during which data was collected. All three of the datasets contain more than 2.5 million records, making them far larger than the datasets on which time-aware ItemKNN models have been evaluated before. The news dataset spans only a week, but due to the typical dramatic drift in news, this should be plenty of data to showcase the usefulness of the decay functions. Retail datasets have been collected over multiple months or years, allowing us to inspect the decay functions when the drift is slow or almost non-existent.

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

5.4.2 Algorithms

In our experiments, we compare the five time-aware ItemKNN algorithms we identified in the literature [33, 76, 81, 83, 126], the optimal version of our unified algorithm (UTARS ItemKNN), and the best-performing personalised baselines presented in Verachtert et al. [127]: Sequential Rules(SR) [85, 86], GRU4Rec [134] and $EASE^R$ [121], as well as the traditional ItemKNN algorithm [113, 32].

A detailed description of these algorithms can be found in Chapter 2, Section 2.5.

5.4.3 Evaluation metric(s)

To evaluate each algorithm's quality, we compute their Normalised Discounted Cumulative Gain and (Calibrated) Recall on the top-10 recommendations (NDCG@10 and Recall@10) [115, 80].

Normalised Discounted Cumulative Gain (NDCG) Ranking metric, computed on a ranked top-k list, that gives lower values to hits lower down in the ranking of recommendations.

Calibrated Recall Accuracy metric, which computes the portion of the targets that have been recovered for each user by the recommender system. We use the calibrated variant proposed by Steck [121] instead of traditional Recall, to avoid unfair low scores for users with more than k targets when evaluating a top-k list (r_u).

For the full description of each metric, see Chapter 2, Section 2.7.

5.4.4 Evaluation Scenarios

To investigate the performance of the algorithms in different settings, we look at two different evaluation scenarios. First, we use a timed-last-item scenario, which is a variation of the traditional last-item splitting technique, which guarantees that training and evaluation datasets do not overlap in time, thus avoiding leakage [65]. The second scenario we evaluate is a purely time-based split, where all of a user's visits after a timestamp are the target for prediction.

We presented the details of these scenarios in Chapter 2, Section 2.4.

The timed-leave-last-one-out scenario evaluates models on their ability to predict a single item at the end of a user session, simulating recommendations on an item page. The timed scenario focuses on retrieving all users' interests given the historic data. This scenario is more closely related to home-page recommendations. Users return and don't always have recent interactions, yet we still want to suggest them relevant articles or products.

5.4.5 Parameter Optimisation

Non-time-aware baselines and cooc-decay methods were optimised using `hyperopt`[3] [19], following the methodology from Verachtert et al. [127]. Rather than using a random search, this library uses a Tree-structure Parzen Estimator [18], which optimises the explored parameters to promising areas of the parameter space.

To find the optimal UTARS settings, we performed a grid search over the various decay functions Γ and Γ', decay parameters α and similarity measures s. While this takes much longer than optimisation using hyperopt, we can be sure that all combinations are tried, and thus better evaluate the quality of our framework.

When optimising the choice of Γ and Γ' for the UTARS ItemKNN model, we only allow configurations where the same decay type is used during training and prediction. The α parameter can differ between training and prediction though. Making this simplification was necessary to avoid exponential growth of the parameter space during optimisation.

The parameters for the original TARS ItemKNN models are similarly optimised via grid search. The full grids can be found in our public repository[4].

For none of the time-aware ItemKNN algorithms do we optimise δ, the amount of training data, as suggested by the experimental setup we follow [127]. We expect the decay function to give the old data an appropriate weight, rather than requiring removal.

5.5 Results

5.5.1 Run Time

In Table 5.3, we present how long it takes to train each algorithm on the various datasets. This training time is an important property when deciding which algorithm to use in a production setting. Algorithms that require long training times introduce a larger computational cost, that needs to be offset by significant improvements in recommendation quality.

We train the models on a google cloud instance, with 8 cores and 50 GB of RAM, for the gradient descent algorithm (GRU4Rec), an NVIDIA T4 GPU is used to speed up training. We find that $EASE^R$ fails to train a model on the Yoochoose and Amazon Toys & Games datasets due to a shortage of memory. The large amount of items in these two datasets, 44 thousand and 69 thousand respectively, causes the dense similarity matrix to become too large to keep in memory. The ItemKNN methods avoid these issues, by constructing a sparse similarity matrix, only keeping the K most similar items for each (center) item.

ItemKNN and the time-decay variants perform similarly, training their model in just a few seconds. Sequential Rules and EASE take longer, but still usually remain under 15 minutes. The cooc-decay ItemKNN variants perform much worse comparatively and take up to 100 times more training time, than the time-decay methods. In the next two

[3] https://hyperopt.github.io/hyperopt/

[4] `https://anonymous.4open.science/r/TARSItemKNN-7A0A/algorithm_config.py`

	Adressa	**Cosmetics-Shop**	**Yoochoose**	**Amazon Games**	**Amazon Toys & Games**
Ding 2005	2	20	17	2	9
Lee 2008	55	136	413	11	45
Liu 2010	2	13	15	1	2
Liu 2012	2	19	20	2	10
Vaz 2013	2	17	17	1	4
UTARS ItemKNN	2	14	15	1	2
ItemKNN	1	14	9	2	9
EASE	2	814	-	200	-
SR	107	365	1 029	18	102
GRU4Rec	1 350	16 119	18 150	379	3 369
Hermann 2010	243	5 371	20 728	303	5 174
Xia 2010	98	1 976	8 556	129	2 034

Table 5.3: Training times for each of the algorithms.

	Γ	Γ'	s
Adressa	exponential, $\alpha = \frac{1}{3*3600}$	exponential, $\alpha = 0$	conditional prob.
Cosmeticsshop	exponential, $\alpha = 0$	exponential, $\alpha = \frac{1}{3600}$	cosine similarity
Yoochoose	exponential, $\alpha = \frac{1}{14*24*3600}$	exponential, $\alpha = \frac{1}{3600}$	cosine similarity
Amazon Games	linear, $\alpha = 5$	linear, $\alpha = 1$	conditional prob.
Amazon Toys & Games	exponential, $\alpha = 0$	exponential, $\alpha = \frac{1}{12 \cdot 3600}$	cosine similarity

Table 5.4: Optimal parameters for the UTARS ItemKNN algorithm in leave-last-one-out scenario.

sub-sections, which present the evaluation of the quality of the recommendations, we also see that neither of these two algorithms makes up for the long training time by outperforming the other algorithms in recommendation quality. GRU4Rec suffers from similarly long training times, despite the availability of a GPU.

Due to the long training time of the cooc-decay methods on the Yoochoose dataset, we decided to exclude these combinations from our experiments. We have sufficient evidence from the other datasets to analyse their performance.

5.5.2 Leave-last-one-out scenario

The optimal settings for UTARS ItemKNN can be found in Table 5.4. For Adressa and Amazon Games, we see that conditional probability is the best similarity metric, whereas for the other retail datasets, it is the cosine similarity that performs best. Given the known popularity bias in Adressa, it is likely that the conditional probability's bias towards recommending these popular articles gives it an edge in the evaluation. For Amazon Games, there is a similar popularity bias for the recently popular games. The decay parameters for Adressa, CosmeticsShop and Yoochoose are aligned with the optimal δ values found in Verachtert et al. [127]. For CosmeticsShop, none of the interactions is given a lower weight, as no decay is applied during training. With Yoochoose, we see that a half-life of two weeks performs optimally. This is clearly linked to the experiments of Verachtert et al. [127] which found an optimal performance only using the most recent 10 days for training non-time-aware models. The news dataset Adressa has an even shorter optimal half-life of three hours, quickly reducing the relevance of older interactions as the more recent ones are more important to recommend the immediate future. During prediction, we do see an opposite pattern, Adressa histories are not decayed, thus solely relying on the decay during prediction, to reduce the importance of less relevant historical items. For the CosmeticsShop and Yoochoose datasets, the optimal parameters conversely both have a strong decay ($\alpha = \frac{1}{3600}$), when predicting, we are best to give only the user's very recent interactions a high weight. These results clearly show that it is important to decouple the decay parameters from training and prediction since they serve different purposes. During training, the decay indicates the relevance of an event to find similarities with other items, while during prediction, the decay matches the interactions' weights to their relevance for the intent of the user. On Amazon Games, we see that the optimal decay is a linear decay, with the steepest decay ($\alpha = 5$) during training, this indicates that the dataset's most relevant training data is the most recent data. This makes sense, usually the most recent games are the ones that users are most likely to purchase and review. On Amazon Toys & Games we see optimal settings that are similar to CosmeticsShop and Yoochoose, only Γ', the prediction decay function, is slightly shallower, but given the longer period over which the dataset has been collected, the resulting decay for most interactions is roughly the same.

Comparing the algorithm results in Table 5.5 and 5.6, we note that the optimal settings on most retail datasets for UTARS correspond to the algorithm proposed by Liu et al. [81], exponential decay with cosine similarity. The Sequential Rules algorithm outperforms UTARS on CosmeticsShop and Yoochoose datasets in this scenario. On both Amazon datasets SR is outperformed by the UTARS ItemKNN algorithm, indicating that the reviews contain less sequential information, though on Amazon Games EASE^{R} trained on recent data improves on UTARS ItemKNN.

For Adressa, the use of a conditional probability similarity function and exponential decay result in a dramatic uplift compared to the other algorithms. Thanks to these settings, the algorithm manages to recommend recently popular items to the right people, strongly improving over the other baselines.

	Adressa	**Cosmetics-Shop**	**Yoochoose**	**Amazon Games**	**Amazon Toys & Games**
Ding 2005	2.55	6.43	16.89	3.26	**3.50**
Lee 2008	1.40	5.01	17.10	3.06	2.85
Liu 2010	3.81	6.43	18.53	3.63	**3.50**
Liu 2012	0.60	4.87	16.59	3.17	2.84
Vaz 2013	2.15	5.89	17.16	1.74	1.59
GRU4Rec	3.87	3.30	13.61	1.54	0.50
$EASE^R$	8.69	4.60	-	**4.03**	-
SR	4.53	**7.23**	**20.69**	3.41	3.11
ItemKNN	5.40	4.90	17.84	3.31	2.86
UTARS ItemKNN	**12.05**	6.43	18.53	3.95	**3.50**
Hermann 2010	1.78	0.0	-	0.53	0.02
Xia 2010	0.44	0.29	-	1.58	1.04

Table 5.5: NDCG@10 results for the leave-last-one-out scenario. EASE could not be run on the Yoochoose dataset due to memory constraints.

	Adressa	**Cosmetics-Shop**	**Yoochoose**	**Amazon Games**	**Amazon Toys & Games**
Ding 2005	4.13	10.89	28.78	5.50	**5.02**
Lee 2008	2.65	8.90	29.39	5.11	4.26
Liu 2010	7.19	10.89	31.21	5.75	**5.02**
Liu 2012	1.14	8.56	28.40	5.26	4.21
Vaz2013	4.37	9.88	28.96	2.55	2.43
GRU4Rec	7.60	5.40	23.88	3.24	0.79
$EASE^R$	13.46	7.98	-	**6.73**	-
SR	9.15	**12.00**	**32.61**	5.60	4.33
ItemKNN	10.63	8.70	30.26	5.65	4.30
UTARS ItemKNN	**20.24**	10.89	31.21	6.63	**5.02**
Hermann 2010	3.32	0.01	-	1.23	0.04
Xia 2010	0.73	0.52	-	3.29	1.92

Table 5.6: Recall@10 results for the leave-last-one-out scenario. EASE could not be run on the Yoochoose dataset due to memory constraints.

	Γ	Γ'	s
Adressa	exponential, $\alpha = \frac{1}{3600}$	exponential, $\alpha = \frac{1}{12 \cdot 3600}$	conditional prob.
Cosmeticsshop	exponential, $\alpha = 0$	exponential, $\alpha = 0$	conditional prob.
Yoochoose	concave, $\alpha = 0.8$	concave, $\alpha = 0.5$	cosine similarity
Amazon Games	linear, $\alpha = 5$	linear, $\alpha = 5$	conditional prob.
Amazon Toys & Games	linear, $\alpha = 1$	linear, $\alpha = 5$	conditional prob.

Table 5.7: Optimal parameters for the UTARS ItemKNN algorithm in the timed-split scenario.

5.5.3 Timed split scenario

Comparing the optimal parameters for the timed scenario with the optimal parameters for the leave-last-one-out scenario, we immediately see that exponential decay is less prevalent than in the last-item scenario. On CosmeticsShop, disabling decay performs optimally, while on Yoochoose the concave decay performs the best. Cosine similarity is also outperformed by conditional probability on the CosmeticsShop dataset. For Adressa we notice an even steeper training decay as well as now decaying user histories as well. Given that the two tasks are significantly different, we see that through parameter tuning we can adapt the model to be more suited for the task at hand. Contrasting to the strong prediction decays for the leave-last-one-out scenario, we see here that putting equal weight on historic interactions allows for a more personalised experience in the retail datasets.

The results for the timed split scenario are found in Tables 5.8 and 5.9. Our unified framework outperforms or matches the other TARS ItemKNN models, and it is the best-performing algorithm on both Amazon Datasets. SR still outperforms the other methods on CosmeticsShop and Yoochoose datasets but is outperformed by the UTARS ItemKNN method on the Adressa dataset. The strong performance of Sequential Rules even on the timed scenario is surprising, given that this algorithm only uses a user's last seen item to generate recommendations. We surmise that the algorithm is able to perform so well on these retail datasets by correctly predicting items that often occur close to the user's last item. On the Yoochoose dataset, also ItemKNN baseline outperforms the UTARS method. Note that this method was trained on an optimised window of data, indicating that none of our decay functions was able to correctly account for the type of drift present in this dataset for this particular scenario. Concave decay was the optimal decay of the ones we investigated, but it seems to be not flexible enough to give enough events equal weight before strongly discounting older interactions. A stepwise or sigmoid decay might be worth looking into for this type of scenarios. As in the first scenario, the cooc-decay methods are clearly outperformed. Despite the intuition that we can find better similarities when considering the time between events, this does not translate to better performance on any of the datasets.

	Adressa	Cosmetics-Shop	Yoochoose	Amazon Games	Amazon Toys & Games
Ding 2005	1.53	4.28	9.18	2.46	1.07
Lee 2008	0.93	3.48	8.37	1.71	0.76
Liu 2010	3.04	3.49	8.81	1.80	0.77
Liu 2012	0.46	3.42	8.68	1.68	0.75
Vaz 2013	1.50	3.05	6.51	0.42	0.33
GRU4Rec	9.55	2.30	5.51	1.12	0.21
$EASE^R$	6.26	3.98	-	2.07	-
SR	6.32	**7.25**	**18.52**	1.81	0.60
ItemKNN	7.09	4.29	17.86	1.78	1.01
UTARS ItemKNN	**18.48**	4.46	10.73	**2.94**	**1.08**
Hermann 2010	1.50	0.09	-	0.53	0.01
Xia 2010	0.43	0.46	-	1.50	0.10

Table 5.8: NDCG@10 results for the timed split scenario. EASE could not be run on the Yoochoose and Amazon Toys and Games datasets due to memory constraints. The approaches proposed by Hermann [51] and Xia et al. [135] were not usable on the CosmeticsShop and Yoochoose datasets due to run-time constraints.

	Adressa	Cosmetics-Shop	Yoochoose	Amazon Games	Amazon Toys & Games
Ding 2005	2.02	5.56	13.36	3.71	1.55
Lee 2008	1.46	4.60	14.19	2.78	1.17
Liu 2010	4.64	4.57	13.55	3.02	1.19
Liu 2012	0.74	4.50	12.94	2.66	1.17
Vaz 2013	2.52	3.97	7.94	0.86	0.55
GRU4Rec	19.19	2.86	2.50	2.09	0.34
$EASE^R$	10.35	5.17	-	3.30	-
SR	11.24	**11.97**	**32.69**	3.06	0.85
ItemKNN	11.10	6.32	30.28	2.80	1.59
UTARS ItemKNN	**24.38**	5.72	15.05	**4.88**	**1.72**
Hermann 2010	3.06	0.01	-	0.53	0.01
Xia 2010	0.89	0.53	-	1.50	0.08

Table 5.9: Recall@10 results for the timed split scenario. EASE could not be run on the Yoochoose and Amazon Toys and Games datasets due to memory constraints. The approaches proposed by Hermann [51] and Xia et al. [135] were not usable on the CosmeticsShop and Yoochoose datasets due to run-time constraints.

5.6 Conclusion

We have shown that the unified framework we propose helped to find settings that improve on the known time-aware ItemKNN variants. The parameters can be tuned to different tasks through optimisation, e.g., next-item-in-session prediction or homepage recommendations. Thanks to the flexibility of the framework, the algorithm is competitive on all datasets, usually ranking first or second among the methods tested. As such, Time-Aware ItemKNN models are competitive with other baselines in the evaluation of time-aware algorithms.

Chapter 6

The Impact of a Popularity Punishing Parameter on ItemKNN Recommendation Performance

Collaborative filtering techniques have a tendency to amplify popularity biases present in the training data if no countermeasures are taken. The ItemKNN algorithm with conditional probability-inspired similarity function has a hyperparameter α that allows one to counteract this popularity bias. In this chapter, we perform a deep dive into the effects of this hyperparameter in both online and offline experiments, with regard to both accuracy metrics and equality of exposure. Our experiments show that the hyperparameter can indeed counteract popularity bias in a dataset. We also find that there exists a trade-off between countering popularity bias and the quality of the recommendations: Reducing popularity bias too much results in a decrease in click-through rate, but some counteracting of popularity bias is required for optimal online performance. [1]

[1]Chapter based on "The Impact of a Popularity Punishing Parameter on ItemKNN Recommendation Performance". Robin Verachtert, Lien Michiels, Jeroen Craps and Bart Goethals. Accepted at the European Conference on Information Retrieval 2023.

6.1 Introduction

Collaborative filtering algorithms are widely used for recommendation systems. To make predictions of what users may like, they rely on past preferences for items expressed by users. These preferences can, for example, be expressed by interacting with an item. Collaborative filtering methods can suffer from a 'rich get richer' effect when they fail to address the popularity bias in the data. For example, when some items are visited more often by users, the recommendation algorithm is also more likely to recommend them. This bias towards already popular items is generally considered undesirable, and many solutions have been proposed to address this bias [e.g. 1, 89, 143]. Even some of the earlier works on collaborative filtering were mindful of this inherent popularity bias. When Deshpande and Karypis [32] proposed the ItemKNN algorithm, they added a hyperparameter α to their conditional probability-inspired similarity function with the explicit purpose of discounting popular items that may otherwise dominate recommendations. Recent works have shown that despite advances in the field, ItemKNN and other nearest neighbour-based methods are still competitive, provided they are well-tuned [38, 37, 86, 127]. Because of their inherent scalability, they remain popular methods in production environments.

In this chapter, we investigate how different values of the hyperparameter α impact performance and equality of exposure, as a measure of popularity bias, in both offline and online experiments with ItemKNN on three news datasets.

We answer the following three research questions:

- **RQ1:** How does the hyperparameter α impact the equality of exposure?
- **RQ2:** How does the hyperparameter α impact accurracy and CTR results?
- **RQ3:** Do the offline and online results agree?

Our work is done in the context of the popular item-to-item recommendation paradigm, recommending similar items in the context of another item, which we will refer to as *context item*. We focus our work on the news domain, as they have a specific interest in combatting popularity bias for ethical reasons, and, of course, because our partners agreed to perform the online tests discussed in this chapter. All data processed in these experiments was collected in accordance with GDPR: Users consented to receive personalised recommendations, as well as to have their data analysed and to participate in AB testing.

We find that the hyperparameter α can be used to increase the equality of exposure. Secondly, we find that it is necessary to seek a trade-off between equality of exposure and recommendation quality. We leave a thorough investigation into this trade-off for future work. Finally, we note that our offline and online results do not align due to the inherent popularity bias persisted in the offline evaluation [16].

6.2 Related Work

Popularity bias has been extensively studied in the context of recommender systems [e.g. 1, 89, 143]. Although the effect of popularity bias on ItemKNN has been studied [3], to the best of our knowledge, the impact of the hyperparameter α on popularity bias has not. In the original work by Deshpande and Karypis [32], the impact of α is evaluated solely in terms of MRR and HitRate, both accuracy measures. Recent work by Pellegrini et al. [99] suggests that not recommending popular items makes recommendations more personalised and can positively impact the recommender system's performance.

ItemKNN remains a popular and competitive baseline, despite recent advances in recommendation algorithms [38, 37, 86, 127].
Due to their scalability, neighbourhood-based methods such as ItemKNN remain a popular choice in production settings [12, 34, 71, 106]. Therefore, a thorough investigation of how the popularity bias can be countered is of great practical relevance.

Offline and online results often do not correlate [16, 41, 110], although some works have achieved success [45, 90]. Popularity bias is an important factor in this failure to correlate and thus we investigate its impact in this chapter[16].

6.3 Experimental Setup

In this chapter, we focus on the item-to-item recommendation problem. The recommendation system needs to recommend users new items while they are currently visiting an item page on the website. The item the user is visiting is the only information the system uses to generate recommendations.

The dataset $\mathcal{D}$ consists of triplets (u, i, t) where $u \in U$ is the user, $i \in I$ is the item, and $t \in \mathbb{N}$ is the timestamp of when user u interacted with item i. Then the recommendation for user u is a function: $\phi(\mathcal{D}_u^l)$, where $\mathcal{D}_u$ is the list of items that the user has seen and $\mathcal{D}_u^l$ is the last item that the user has seen.

6.3.1 Algorithm

We use the ItemKNN algorithm, with the similarity between items computed using the conditional probability-inspired similarity function, defined as

$$sim(i, j) = \frac{|\{u|i, j \in \mathcal{D}_u\}|}{|\{u|i \in \mathcal{D}_u\}| \cdot |\{u|j \in \mathcal{D}_u\}|^\alpha}$$

Here, i is a context item, j is a target item and α is a hyperparameter that punishes popular items in the similarity computation [32].
Specific values for α can be linked to other similarity measures. When $\alpha = 1$ it provides

Website	Users (per day)	Articles read (per day)	Clicks (per day)	$\|U\|$	$\|I\|$	$\|\mathcal{D}\|$	Gini coeff.
NP1	300K	1M	25K	410 843	2 382	4 049 944	0.79
NP2	200K	800K	14K	234 839	2 404	2 852 956	0.77
NP3	1M	4M	160K	1 215 900	5 531	13 842 991	0.88

Table 6.1: Statistics of websites used in the online tests.

the same recommendations as the lift similarity measure. In the specific case of item-to-item recommendations, $\alpha = 0.5$ leads to the same recommendations as cosine similarity.

6.3.2 Metrics

To evaluate the exposure of articles, we measure both the item-space coverage and the Gini coefficient as suggested in previous works on evaluation [47, 26]. Coverage computes the percentage of the available catalogue recommended at least once during an experiment, while the Gini coefficient gives more insight into the recommendation distribution by measuring the inequalities in the number of recommendations each item in the catalogue receives. To evaluate the accuracy of the recommendations, we measured normalised discounted cumulative gain (NDCG) [59], recall [47] and mean reciprocal rank [47]. For brevity, we report only the NDCG results in this chapter. Both other accuracy metrics support the same findings. In online trials, we evaluate the quality of the recommendations by click-through rate (CTR).

6.3.3 Datasets

For our experiments, we use three different newspaper websites as our testing platforms, referred to as NP1, NP2 and NP3. The statistics of online traffic and offline exports on these websites can be found in Table 6.1. Offline datasets are constructed by selecting events from an eight-day window on the website.

6.3.4 Offline experiments

In our offline experiments, we closely mimic the online setup. The first day of our eight-day dataset is used to make sure that we always have a full day of training data when training a model. The second day is used for optimising other hyperparameters than α. The last six days are used for evaluation. Models are trained, following the online setting, on a single day of training data. During optimisation and evaluation, we expect the model to predict a user's last event between 10 AM and 2 PM on each day, using their second to last event in that window as the context item. The measurements from each of the six evaluation days are averaged and reported in this chapter.

As our online tests show three items to the user, we also evaluate the offline metrics on the top three recommendations.

	Coverage@3 (%)					Gini coeff.				
α	0.0	0.2	0.5	0.7	1.0	0.0	0.2	0.5	0.7	1.0
NP1	71	87	94	**97**	95	0.91	0.89	0.83	0.79	**0.76**
NP2	57	78	93	**94**	**94**	0.92	0.90	0.83	0.78	**0.76**
NP3	78	94	97	**100**	99	0.91	0.90	0.80	**0.70**	**0.70**

Table 6.2: Coverage and Gini coefficient results for each of the hyperparameter configurations in the online experiments.

We ran our experiments for $\alpha \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$. For our online tests, we selected $\alpha \in \{0, 0.2, 0.5, 0.7, 1\}$, as they resulted in different exposure distributions. For brevity, we only report results for these values of α.

6.3.5 Online Experiments

Recommendations were displayed in a horizontal list of three items, just after the end of an article. The models for both the control and treatment groups are re-trained every 15 minutes, using a day of training data, following the results from Chapter 4. In order to evaluate the impact of α in a real and dynamic environment, we performed a sequence of trials. In each of these trials, a control group of 75% of the users received recommendations using $\alpha = 0.5$. The treatment group (25% of users) received recommendations using a different $\alpha \in \{0, 0.2, 0.7, 1\}$ for each trial period. As it is not possible to compare the CTR between treatment groups, we instead use the lift in CTR for each treatment group compared to the control group during each trial.

6.4 Experiments

6.4.1 RQ1: How Does the Hyperparameter α Impact the Equality of Exposure?

In Table 6.2 we show that increasing α leads to higher coverage and to more equal exposure between items. Increasing α from 0.7 to 1.0 does lead to only minor improvements in the Gini coefficient and to a reduction of coverage in two datasets.
In Figure 6.1 we look beyond the metrics and inspect how the α hyperparameter impacts how often items are recommended on the NP3 website. Items are sorted by popularity, from most popular to least popular along the x-axis. When α is 0, almost all recommendations are from the most popular items. As the value of the hyperparameter increases, more and more different items are recommended, until the distribution shifts when α is 1, and mostly unpopular items are recommended. This insight explains the slight decrease in coverage for some of the datasets, and why the Gini coefficient did not decrease further when increasing α to the max. These distribution plots, also show that none of the α settings provides true equality of exposure, as the middle section of items is always under-recommended, compared to popular or unpopular items depending on the value of α.

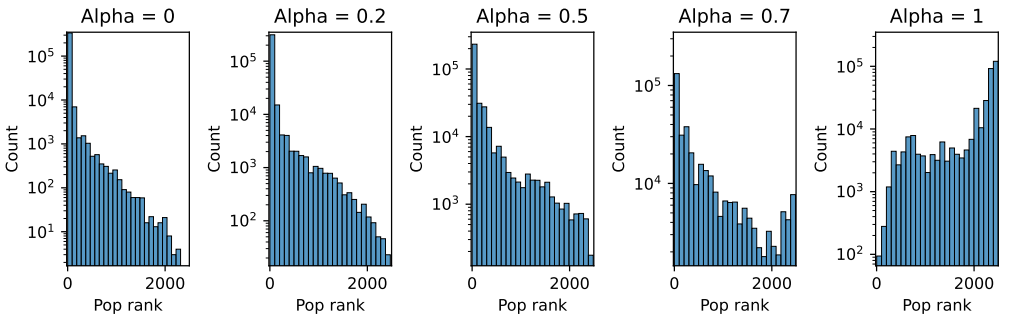

Figure 6.1: Number of times items are recommended on the NP3 website experiment, ranked by popularity. The lowest rank is the most visited item.

	NDCG@3 (%)					CTR lift (%)				
α	0.0	0.2	0.5	0.7	1.0	0.0	0.2	0.5	0.7	1.0
NP1	8.52	**9.15**	7.68	5.27	0.70	-6.40	-4.05	**0**	-4.82	-21.26
NP2	5.54	**6.43**	6.41	4.47	1.07	-3.28	-1.28	**0**	-6.12	-26.45
NP3	6.44	**7.02**	6.48	4.16	0.40	-6.87	-3.91	**0**	-6.93	-31.60

Table 6.3: NDCG@3 (offline) results and CTR (online) results. CTR results are relative performance compared to the control setting ($\alpha = 0.5$).

6.4.2 RQ2: How Does the Hyperparameter α Impact Accurracy and CTR Results?

In Table 6.3, we show the NDCG@3 for each of the settings of α in our offline tests and the lift in CTR during the online tests.
In the offline experiments increasing the α hyperparameter beyond 0.2 leads to a decrease in performance. As less popular items are recommended, accuracy suffers. Online we find a similar result, higher values of α do not correlate with a higher CTR. However, maximal online performance is reached with the control setting of $\alpha = 0.5$.

So, while a higher α results in a higher coverage and a lower Gini coefficient, both the click-through rate and the NDCG show a decrease in performance when we increase α too much. In our news use-cases, exposure equality and countering popularity bias need to be balanced with recommendation performance. Popular items are relevant to many users, and so if we want to showcase more, less popular, items, we might need to accept a performance decline.

6.4.3 RQ3: Do the Offline and Online Results Agree?

In the offline results, the optimal setting for all datasets is $\alpha = 0.2$. However, in our online results, $\alpha = 0.2$ is not optimal, instead $\alpha = 0.5$ is the optimal setting.

Our datasets, like many news datasets, show an unbalanced reading behaviour, indicated

by the high Gini coefficient in Table 6.1. Users read the most popular items much more often than the other items. This popularity bias leads to higher performance in offline results for algorithms with more popularity bias (lower α). However, in the production setting, recommending mostly popular items leads to recommending popular items not related to the context item. Users looking for related articles do not click on these popularity-based recommendations. These results follow the common finding, due to popularity bias offline and online results do not align nicely. However, we can see the value of the offline experimentation in the performance of the $\alpha = 1$ setting. The bad offline performance is reflected in the online results.

6.5 Conclusion

We find that while the hyperparameter α is able to counteract popularity bias, it is only a proxy for true exposure equality. Therefore, further research is required on how to combat the popularity bias of the ItemKNN algorithm. Secondly, we note that our offline and online results do not align, due to the inherent popularity bias in typical offline evaluation [16, 15, 139]. Our findings suggest that it is worthwhile to opt for suboptimal offline test results in terms of accuracy, but with a lower Gini index. However, a trade-off should be sought between fair exposure and user experience. We leave a thorough investigation of this trade-off and a framework for determining the setting most likely to perform best in online tests for future work. Finally, we note that our results are limited to the news domain. We see no reason to believe that our findings will not generalize to other domains, as they were not dependent on specific characteristics of the news context.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

When we leave the controlled environment of experimentation and use recommender systems in production environments, we encounter additional complexities, and known problems become more difficult. One particularly interesting complication is the effect of temporal dynamics. Recommendation systems by their very nature are exposed to constantly changing environments and users. In this thesis, we investigated ways to mitigate and exploit the impact of temporal dynamics on recommendation performance. We found that ignoring the temporal dynamics results in poor performance, regardless of the algorithm's quality.

The presence of change in real-world applications forces practitioners to reason about updating models, and when to do so. We find that scheduling the model updates based on the amount of information collected since the last update is a more efficient method than using the commonly used regular interval-based method. The smart scheduling approaches maintain performance, while being significantly cheaper, or increase performance for the same cost, depending on the availability of resources.

Further, we found that when computing models, it is vital to reason about how much data is used to train the model. An immediate result of using less data is that it reduces training time and costs, both of which are important functional requirements for a production environment. The less obvious finding is that using less data also improves performance drastically for many methods. This increase causes simple baseline methods like ItemKNN and Popularity to outperform more complex deep learning methods when used in dynamic environments. By using only recent data, we can teach these models with limited capacity, about only the relevant information from the recent past.

The ideal situation, is obviously that the model can choose which events are relevant and which are not. We addressed this desire by proposing a framework for time-aware item-based nearest neighbour methods, which apply a decay function before computing similarities. This approach still follows the heuristic that older data is less relevant but keeps it in the model to provide additional information. We find that this indeed improves normal ItemKNN methods, which are only trained on the most recent data.

Finally, we investigated how we can avoid popularity bias in the recommendations, and

the impact this has on users' interactions with the recommendations. We performed an extensive offline and online experiment that compared the impact we can have on coverage, popularity bias, and recommendation quality when we punish popular items in an item-based nearest neighbour approach. We find that punishing popular items indeed improves recommendation quality, click-through rates, as well as the fairness of exposure, though overdoing it results in catastrophic performance degradation.

7.2 Future Work

Long Term Interests In our work, we have focused on the "short-term" interests and exploited recent interactions to predict the immediate next visited item(s). The goal of personalisation, though, should be to also use the longer-term history of each user, to provide an even more unique experience tailored to each individual. Of particular interest in this area, is the recommendation of niche items to the users that are interested in this topic. The challenge comes from the fact that these niche interests are usually less represented in the dataset than popular topics, they have fewer articles, and finally, there is usually more time between publications of the articles. All of these factors introduce an extremely sparse pattern that is hard to surface by traditional collaborative filtering means. Specific methods to look for these sparse patterns can help news publishers get the right news to the right people.

Efficient algorithms Most companies that use recommender systems have a tight budget to use to serve and train their recommendation models. So for them, the ability to efficiently compute small models that still get high quality is an important asset. Recent advances in deep learning often result in algorithms that are high in resources, training time, and model size. Further research into algorithms that are both efficient to compute and can be served with minimal resources will make advances available to small- and medium-sized companies as well as the tech giants that are mostly served right now.

Scheduling Our scheduling systems were based on heuristics, which showed a correlation with better scheduling moments. Approaching the scheduling problem from a control systems angle could help further improve scheduling, and allow an even better return on investment for companies in their recommendation models. We considered, but never completed avenues using reinforcement learning (RL) and counterfactual reasoning, both of which seem suitable for application to the scheduling problem. Using RL, the agent would have to decide on one of two actions (update or no update), and the reward function is a combination of cost (negative) and performance (positive). Counterfactual reasoning, then, would try to answer the question of what would happen if we did schedule an update, or not.

Bibliography

[1] Himan Abdollahpouri, Robin Burke, and Bamshad Mobasher. Controlling Popularity Bias in Learning-to-Rank Recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, page 42–46, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346528. doi: 10.1145/3109859.3109912. 90, 91

[2] Himan Abdollahpouri, Robin Burke, and Bamshad Mobasher. Controlling Popularity Bias in Learning-to-Rank Recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, page 42–46, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346528. doi: 10.1145/3109859.3109912. URL https://doi.org/10.1145/3109859.3109912. 5

[3] Himan Abdollahpouri, Masoud Mansoury, Robin Burke, and Bamshad Mobasher. The Unfairness of Popularity Bias in Recommendation. *CEUR Workshop Proceedings*, 2440, 2019. URL https://ceur-ws.org/Vol-2440/paper4.pdf. 91

[4] Charu C. Aggarwal. *Recommender Systems: The Textbook*. Springer International Publishing, Cham, 2016. ISBN 978-3-319-29659-3. doi: 10.1007/978-3-319-29659-3. URL https://doi.org/10.1007/978-3-319-29659-3. 13, 29, 32

[5] Marie Al-Ghossein, Pierre-Alexandre Murena, Talel Abdessalem, Anthony Barré, and Antoine Cornuéjols. Adaptive collaborative topic modeling for online recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, page 338–346, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450359016. doi: 10.1145/3240323.3240363. URL https://doi.org/10.1145/3240323.3240363. 40, 43

[6] Vito Walter Anelli, Tommaso Di Noia, Eugenio Di Sciascio, Azzurra Ragone, and Joseph Trotta. Local popularity and time in top-n recommendation. In Leif Azzopardi, Benno Stein, Norbert Fuhr, Philipp Mayr, Claudia Hauff, and Djoerd Hiemstra, editors, *Advances in Information Retrieval*, pages 861–868, Cham, 2019. Springer International Publishing. ISBN 978-3-030-15712-8. doi: 10.1007/978-3-030-15712-8_63. 63, 75

[7] Vito Walter Anelli, Alejandro Bellogin, Antonio Ferrara, Daniele Malitesta, Felice Antonio Merra, Claudio Pomo, Francesco Maria Donini, and Tommaso Di Noia. Elliot: A comprehensive and rigorous framework for reproducible recommender systems evaluation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, page 2405–2414, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380379. doi: 10.1145/3404835.3463245. URL https://doi.org/10.1145/3404835.3463245. 12

[8] Susan C. Anyosa, João Vinagre, and Alípio M. Jorge. Incremental matrix co-factorization for recommender systems with implicit feedback. In *Companion Proceedings of the The Web Conference 2018*, WWW '18, page 1413–1418, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee. ISBN 9781450356404. doi: 10.1145/3184558.3191585. URL https://doi.org/10.1145/3184558.3191585. 40

[9] Imad Aouali, Amine Benhalloum, Martin Bompaire, Benjamin Heymann, Olivier Jeunen, David Rohde, Otmane Sakhi, and Flavian Vasile. Offline Evaluation of Reward-Optimizing Recommender Systems: The Case of Simulation, 2022. URL https://arxiv.org/abs/2209.08642. 7

[10] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, nov 1998. ISSN 0004-5411. doi: 10.1145/293347.293348. URL https://doi.org/10.1145/293347.293348. 42

[11] Maryam Aziz, Jesse Anderton, Kevin Jamieson, Alice Wang, Hugues Bouchard, and Javed Aslam. Identifying new podcasts with high general appeal using a pure exploration infinitely-armed bandit strategy. In *Proceedings of the 16th ACM Conference on Recommender Systems*, RecSys '22, page 134–144, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392785. doi: 10.1145/3523227.3546766. URL https://doi.org/10.1145/3523227.3546766. 3

[12] Riccardo Bambini, Paolo Cremonesi, and Roberto Turrin. *A Recommender System for an IPTV Service Provider: a Real Large-Scale Production Environment*, pages 299–331. Springer US, Boston, MA, 2011. ISBN 978-0-387-85820-3. doi: 10.1007/978-0-387-85820-3_9. 91

[13] Immanuel Bayer, Xiangnan He, Bhargav Kanagal, and Steffen Rendle. A generic coordinate descent framework for learning from implicit feedback. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 1341–1350, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee. ISBN 9781450349130. doi: 10.1145/3038912.3052694. URL https://doi.org/10.1145/3038912.3052694. 19, 62, 64

[14] Joeran Beel and Victor Brunel. Data pruning in recommender systems research: Best-practice or malpractice? In *13th ACM Conference on Recommender Systems (RecSys)*, 2019. 47

[15] Joeran Beel and Stefan Langer. A Comparison of Offline Evaluations, Online Evaluations, and User Studies in the Context of Research-Paper Recommender Systems. In Sarantos Kapidakis, Cezary Mazurek, and Marcin Werla, editors, *Research and Advanced Technology for Digital Libraries*, pages 153–168, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24592-8. doi: 10.1007/978-3-319-24592-8_12. 95

[16] Joeran Beel, Marcel Genzmehr, Stefan Langer, Andreas Nürnberger, and Bela Gipp. A Comparative Analysis of Offline and Online Evaluations and Discussion of Research Paper Recommender System Evaluation. In *Proceedings of the International Workshop on Reproducibility and Replication in Recommender Systems Evaluation*, RepSys '13, page 7–14, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450324656. doi: 10.1145/2532508.2532511. 90, 91, 95

[17] David Ben-Shimon, Alexander Tsikinovsky, Michael Friedmann, Bracha Shapira, Lior Rokach, and Johannes Hoerle. Recsys challenge 2015 and the yoochoose dataset. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, page 357–358, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336925. doi: 10.1145/2792838.2798723. URL https://doi.org/10.1145/2792838.2798723. 15, 64, 79

[18] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. 24, 2011. URL https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf. 66, 81

[19] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 115–123, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL https://proceedings.mlr.press/v28/bergstra13.html. 66, 81

[20] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining (SDM)*, pages 443–448, 2007. doi: 10.1137/1.9781611972771.42. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611972771.42. 42, 43, 63

[21] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3(null):993–1022, March 2003. ISSN 1532-4435. 43

[22] Veronika Bogina, Tsvi Kuflik, Dietmar Jannach, Maria Bielikova, Michal Kompan, and Christoph Trattner. Considering temporal aspects in recommender systems: a survey. *User Modeling and User-Adapted Interaction*, pages 1–39, 2022. doi: 10.1007/s11257-022-09335-w. 62, 63, 75

[23] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D. Sculley. The ml test score: A rubric for ml production readiness and technical debt reduction. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1123–1132, 2017. doi: 10.1109/BigData.2017.8258038. 40, 43, 49

[24] Pedro G. Campos, Fernando Díez, and Iv´n Cantador. Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction*, 24(1-2):67–119, 2014. doi: 10.1007/s11257-012-9136-x. URL https://doi.org/10.1007/s11257-012-9136-x. 21, 40, 63, 75, 77

[25] Rocío Cañamares, Pablo Castells, and Alistair Moffat. Offline evaluation options for recommender systems. *Information Retrieval Journal*, 23(4):387–410, 2020. doi: 10.1007/s10791-020-09371-3. 40

[26] Pablo Castells, Neil Hurley, and Saúl Vargas. *Novelty and Diversity in Recommender Systems*, pages 603–646. Springer US, New York, NY, 2022. ISBN 978-1-0716-2197-4. doi: 10.1007/978-1-0716-2197-4_16. 92

[27] Huiyuan Chen, Yusan Lin, Menghai Pan, Lan Wang, Chin-Chia Michael Yeh, Xiaoting Li, Yan Zheng, Fei Wang, and Hao Yang. Denoising Self-Attentive Sequential Recommendation. In *Proceedings of the 16th ACM Conference on Recommender*

Systems, RecSys '22, page 92–101, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392785. doi: 10.1145/3523227.3546788. URL https://doi.org/10.1145/3523227.3546788. 19

[28] Minmin Chen, Can Xu, Vince Gatto, Devanshu Jain, Aviral Kumar, and Ed Chi. Off-policy actor-critic for recommender systems. In *Proceedings of the 16th ACM Conference on Recommender Systems*, RecSys '22, page 338–349, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392785. doi: 10.1145/3523227.3546758. URL https://doi.org/10.1145/3523227.3546758. 3

[29] Jin Yao Chin, Yile Chen, and Gao Cong. The datasets dilemma: How much do we really know about recommendation datasets? In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, WSDM '22, page 141–149, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391320. doi: 10.1145/3488560.3498519. URL https://doi.org/10.1145/3488560.3498519. 65

[30] Toon De Pessemier, Tom Deryckere, and Luc Martens. Extending the Bayesian Classifier to a Context-Aware Recommender System for Mobile Devices. In *2010 Fifth International Conference on Internet and Web Applications and Services*, pages 242–247, 2010. doi: 10.1109/ICIW.2010.43. 75

[31] Gabriel de Souza Pereira Moreira, Felipe Ferreira, and Adilson Marques da Cunha. News session-based recommendations using deep neural networks. In *Proceedings of the 3rd Workshop on Deep Learning for Recommender Systems*, DLRS 2018, page 15–23, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450366175. doi: 10.1145/3270323.3270328. URL https://doi.org/10.1145/3270323.3270328. 15, 47

[32] Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems*, 22(1):143–177, jan 2004. ISSN 1046-8188. doi: 10.1145/963770.963776. URL https://doi.org/10.1145/963770.963776. 29, 30, 65, 66, 79, 80, 90, 91

[33] Yi Ding and Xue Li. Time weight collaborative filtering. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, CIKM '05, page 485–492, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595931406. doi: 10.1145/1099554.1099689. URL https://doi.org/10.1145/1099554.1099689. 31, 65, 76, 80

[34] Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. Pixie: A System for Recommending 3+ Billion Items to 200+ Million Users in Real-Time. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 1775–1784, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee. ISBN 9781450356398. doi: 10.1145/3178876.3186183. 91

[35] Kim Falk. *Practical Recommender Systems*. Manning, 2019. ISBN 9781617292705. URL https://www.manning.com/books/practical-recommender-systems. 13

[36] Wei Fan. Systematic data selection to mine concept-drifting data streams. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, page 128–137, New York, NY, USA, 2004. Association

for Computing Machinery. ISBN 1581138881. doi: 10.1145/1014052.1014069. URL https://doi.org/10.1145/1014052.1014069. 63

[37] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, page 101–109, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362436. doi: 10.1145/3298689.3347058. URL https://doi.org/10.1145/3298689.3347058. 64, 90, 91

[38] Maurizio Ferrari Dacrema, Simone Boglio, Paolo Cremonesi, and Dietmar Jannach. A troubling analysis of reproducibility and progress in recommender systems research. *ACM Transactions on Information Systems*, 39(2), jan 2021. ISSN 1046-8188. doi: 10.1145/3434185. URL https://doi.org/10.1145/3434185. 64, 74, 90, 91

[39] Cédric Févotte and Jérôme Idier. Algorithms for Nonnegative Matrix Factorization with the -Divergence. *Neural Computation*, 23(9):2421–2456, 09 2011. ISSN 0899-7667. doi: 10.1162/NECO_a_00168. URL https://doi.org/10.1162/NECO_a_00168. 29

[40] João Gama, Indrundefined Žliobaitundefined, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4), mar 2014. ISSN 0360-0300. doi: 10.1145/2523813. URL https://doi.org/10.1145/2523813. 40, 42, 63

[41] Florent Garcin, Boi Faltings, Olivier Donatsch, Ayar Alazzawi, Christophe Bruttin, and Amr Huber. Offline and Online Evaluation of News Recommender Systems at Swissinfo.ch. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, page 169–176, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450326681. doi: 10.1145/2645710.2645745. 91

[42] Diksha Garg, Priyanka Gupta, Pankaj Malhotra, Lovekesh Vig, and Gautam Shroff. Sequence and time aware neighborhood for session-based recommendations: Stan. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, page 1069–1072, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361729. doi: 10.1145/3331184.3331322. URL https://doi.org/10.1145/3331184.3331322. 29, 63, 75

[43] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. Beyond accuracy: Evaluating recommender systems by coverage and serendipity. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, page 257–260, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605589060. doi: 10.1145/1864708.1864761. URL https://doi.org/10.1145/1864708.1864761. 66

[44] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. E-commerce in your inbox: Product recommendations at scale. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, page 1809–1818, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336642. doi: 10.1145/2783258.2788627. URL https://doi.org/10.1145/2783258.2788627. 29

[45] Alois Gruson, Praveen Chandar, Christophe Charbuillet, James McInerney, Samantha Hansen, Damien Tardieu, and Ben Carterette. Offline Evaluation to Make Decisions About Playlist Recommendation Algorithms. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, WSDM '19, page 420–428, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359405. doi: 10.1145/3289600.3291027. 91

[46] Jon Atle Gulla, Lemei Zhang, Peng Liu, Özlem Özgöbek, and Xiaomeng Su. The adressa dataset for news recommendation. In *Proceedings of the International Conference on Web Intelligence*, WI '17, page 1042–1048, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349512. doi: 10.1145/3106426.3109436. URL https://doi.org/10.1145/3106426.3109436. 12, 15, 47, 64, 79

[47] Asela Gunawardana, Guy Shani, and Sivan Yogev. *Evaluating Recommender Systems*, pages 547–601. Springer US, New York, NY, 2022. ISBN 978-1-0716-2197-4. doi: 10.1007/978-1-0716-2197-4_15. 7, 92

[48] Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. Fast Approximate Nearest-Neighbor Search with k-Nearest Neighbor Graph. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, page 1312–1317. AAAI Press, 2011. ISBN 9781577355144. 74

[49] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. 2009. doi: 10.48550/ARXIV.0909.4061. URL https://arxiv.org/abs/0909.4061. 23, 29

[50] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), dec 2015. ISSN 2160-6455. doi: 10.1145/2827872. URL https://doi.org/10.1145/2827872. 12, 13, 15

[51] Christoph Hermann. Time-based recommendations for lecture materials. In Jan Herrington and Craig Montgomerie, editors, *Proceedings of EdMedia + Innovate Learning 2010*, pages 1028–1033, Toronto, Canada, June 2010. Association for the Advancement of Computing in Education (AACE). URL https://www.learntechlib.org/p/34759. 31, 74, 76, 86

[52] Balázs Hidasi and Domonkos Tikk. Fast ALS-Based Tensor Factorization for Context-Aware Recommendation from Implicit Feedback. In Peter A. Flach, Tijl De Bie, and Nello Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 67–82, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-33486-3. doi: 10.1007/978-3-642-33486-3_5. 75

[53] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. 2015. doi: 10.48550/ARXIV.1511.06939. URL https://arxiv.org/abs/1511.06939. 29, 63, 74

[54] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, 2008. doi: 10.1109/ICDM.2008.22. 29

[55] P. Jain and C. S. Dixit. Recommendations with context aware framework using particle swarm optimization and unsupervised learning. *Journal of Intelligent & Fuzzy Systems*, 36(5):4479–4490, 2019. doi: 10.3233/JIFS-179001. 75

[56] Tamas Jambor, Jun Wang, and Neal Lathia. Using control theory for stable and efficient recommender systems. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, page 11–20, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450312295. doi: 10.1145/2187836.2187839. URL https://doi.org/10.1145/2187836.2187839. 43

[57] Dietmar Jannach and Malte Ludewig. When recurrent neural networks meet the neighborhood for session-based recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, page 306–310, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346528. doi: 10.1145/3109859.3109872. URL https://doi.org/10.1145/3109859.3109872. 63, 74, 75

[58] Dietmar Jannach, Malte Ludewig, and Lukas Lerche. Session-based item recommendation in e-commerce: on short-term intents, reminders, trends and discounts. *User Modeling and User-Adapted Interaction*, 27(3):351–392, 2017. 63

[59] Kalervo Järvelin and Jaana Kekäläinen. Cumulated Gain-Based Evaluation of IR Techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, oct 2002. ISSN 1046-8188. doi: 10.1145/582415.582418. 92

[60] Olivier Jeunen. Revisiting offline evaluation for implicit-feedback recommender systems. In *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, page 596–600, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362436. doi: 10.1145/3298689.3347069. URL https://doi.org/10.1145/3298689.3347069. 40

[61] Olivier Jeunen, Koen Verstrepen, and Bart Goethals. Fair offline evaluation methodologies for implicit-feedback recommender systems with MNAR data. In *Proceedings of the ACM RecSys Workshop on Offline Evaluation for Recommender Systems*, REVEAL '18, 2018. 21, 36, 44, 62, 63

[62] Olivier Jeunen, Koen Verstrepen, and Bart Goethals. Efficient similarity computation for collaborative filtering in dynamic environments. In *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, page 251–259, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362436. doi: 10.1145/3298689.3347017. URL https://doi.org/10.1145/3298689.3347017. 40, 42, 75

[63] Olivier Jeunen, Jan Van Balen, and Bart Goethals. Embarrassingly shallow autoencoders for dynamic collaborative filtering. *User Modeling and User-Adapted Interaction*, 34:509–541, 2022. doi: 10.1007/s11257-021-09314-7. 40, 42

[64] Yitong Ji, Aixin Sun, Jie Zhang, and Chenliang 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*, SIGIR '20, page 1749–1752, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380164. doi: 10.1145/3397271.3401233. URL https://doi.org/10.1145/3397271.3401233. 62, 63

[65] Yitong Ji, Aixin Sun, Jie Zhang, and Chenliang Li. A critical study on data leakage in recommender system offline evaluation. *ACM Transactions on Information Systems*, oct 2022. ISSN 1046-8188. doi: 10.1145/3569930. URL https://doi.org/10.1145/3569930. Just Accepted. 19, 23, 64, 80

[66] Ying Jin, Zhuoran Yang, and Zhaoran Wang. Is Pessimism Provably Efficient for Offline RL? In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5084–5096. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/jin21e.html. 75

[67] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 197–206, 2018. doi: 10.1109/ICDM.2018.00035. 13, 19, 62, 64

[68] Wang-Cheng Kang and Julian McAuley. Self-Attentive Sequential Recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 197–206, 2018. doi: 10.1109/ICDM.2018.00035. 19

[69] Michael Kechinov. Cosmeticsshop e-commerce dataset, 2020. URL https://www.kaggle.com/mkechinov/ecommerce-events-history-in-cosmetics-shop. Accessed: 2022-07-26. 15, 64, 79

[70] M. G. Kendall. The treatment of ties in ranking problems. *Biometrika*, 33(3):239–251, 1945. 45, 68

[71] Barrie Kersbergen, Olivier Sprangers, and Sebastian Schelter. Serenade - Low-Latency Session-Based Recommendation in e-Commerce at Scale. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 150–159, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392495. doi: 10.1145/3514221.3517901. 5, 74, 91

[72] Ralf Klinkenberg and Ingrid Renz. Adaptive information filtering: Learning in the presence of concept drifts. *Learning for Text Categorization*, pages 33–40, 1998. 42, 63

[73] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. doi: 10.1109/MC.2009.263. 40

[74] Saurabh Kulkarni and Sunil F. Rodd. Context Aware Recommendation Systems: A Review of the State of the Art Techniques. *Computer Science Review*, 37:100255, 2020. ISSN 1574-0137. doi: https://doi.org/10.1016/j.cosrev.2020.100255. URL https://www.sciencedirect.com/science/article/pii/S1574013719301406. 74, 75

[75] Sara Latifi and Dietmar Jannach. Streaming Session-Based Recommendation: When Graph Neural Networks Meet the Neighborhood. In *Proceedings of the 16th ACM Conference on Recommender Systems*, RecSys '22, page 420–426, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392785. doi: 10.1145/3523227.3548485. 74

[76] Tong Queue Lee, Young Park, and Yong-Tae Park. A time-based approach to effective recommender systems using implicit feedback. *Expert Systems with Applications*, 34(4):3055–3062, 2008. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2007.06.031. URL https://www.sciencedirect.com/science/article/pii/S0957417407002357. 63, 76, 80

[77] Jiacheng Li, Yujie Wang, and Julian McAuley. Time Interval Aware Self-Attention for Sequential Recommendation. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, page 322–330, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368223. doi: 10.1145/3336191.3371786. URL https://doi.org/10.1145/3336191.3371786. 19

[78] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. Neural Attentive Session-Based Recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, page 1419–1428, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349185. doi: 10.1145/3132847.3132926. 74

[79] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. Approximate nearest neighbor search on high dimensional data — experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, 32(8):1475–1488, 2020. doi: 10.1109/TKDE.2019.2909204. 42

[80] Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 689–698, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee. ISBN 9781450356398. doi: 10.1145/3178876.3186150. URL https://doi.org/10.1145/3178876.3186150. 13, 29, 31, 48, 80

[81] Nathan N. Liu, Min Zhao, Evan Xiang, and Qiang Yang. Online evolutionary collaborative filtering. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, page 95–102, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605589060. doi: 10.1145/1864708.1864729. URL https://doi.org/10.1145/1864708.1864729. 29, 31, 63, 65, 76, 80, 83

[82] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. Stamp: Short-term attention/memory priority model for session-based recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '18, page 1831–1839, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3219950. URL https://doi.org/10.1145/3219819.3219950. 63, 74

[83] Yue Liu, Zhe Xu, Binkai Shi, and Bofeng Zhang. Time-based k-nearest neighbor collaborative filtering. In *2012 IEEE 12th International Conference on Computer and Information Technology*, pages 1061–1065, 2012. doi: 10.1109/CIT.2012.217. 31, 63, 76, 80

[84] Corentin Lonjarret, Roch Auburtin, Céline Robardet, and Marc Plantevit. Sequential recommendation with metric models based on frequent sequences. *Data Mining and Knowledge Discovery*, 35(3):1087–1133, 2021. doi: 10.1007/s10618-021-00744-w. 19, 62, 63, 64

[85] Malte. Ludewig and Dietmar. Jannach. Evaluation of session-based recommendation algorithms. *User Modeling and User-Adapted Interaction*, 28(4):331–390, 2018. doi: 10.1007/s11257-018-9209-6. 30, 63, 75, 76, 80

[86] Malte Ludewig, Noemi Mauro, Sara Latifi, and Dietmar Jannach. Performance comparison of neural and non-neural approaches to session-based recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, page 462–466, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362436. doi: 10.1145/3298689.3347041. URL https://doi.org/10.1145/3298689.3347041. 30, 64, 65, 74, 75, 80, 90, 91

[87] Cornelius A. Ludmann. Recommending news articles in the clef news recommendation evaluation lab with the data stream management system odysseus. In *CLEF (Working Notes)*, 2017. 63

[88] Rui Ma, Ning Liu, Jingsong Yuan, Huafeng Yang, and Jiandong Zhang. Caen: A hierarchically attentive evolution network for item-attribute-change-aware recommendation in the growing e-commerce environment. In *Proceedings of the 16th ACM Conference on Recommender Systems*, RecSys '22, page 278–287, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392785. doi: 10.1145/3523227.3546773. URL https://doi.org/10.1145/3523227.3546773. 3

[89] Masoud Mansoury, Himan Abdollahpouri, Mykola Pechenizkiy, Bamshad Mobasher, and Robin Burke. Feedback Loop and Bias Amplification in Recommender Systems. In *Proceedings of the 29th ACM International Conference on Information amp; Knowledge Management*, CIKM '20, page 2145–2148, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368599. doi: 10.1145/3340531.3412152. 90, 91

[90] M. Jeffrey Mei, Cole Zuber, and Yasaman Khazaeni. A Lightweight Transformer for Next-Item Product Recommendation. In *Proceedings of the 16th ACM Conference on Recommender Systems*, RecSys '22, page 546–549, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392785. doi: 10.1145/3523227.3547491. 91

[91] Lien Michiels, Robin Verachtert, and Bart Goethals. Recpack: An(other) experimentation toolkit for top-n recommendation using implicit feedback data. In *Proceedings of the 16th ACM Conference on Recommender Systems*, RecSys '22, page 648–651, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392785. doi: 10.1145/3523227.3551472. URL https://doi.org/10.1145/3523227.3551472. 11

[92] Gabriel De Souza P. Moreira, Dietmar Jannach, and Adilson Marques Da Cunha. Contextual hybrid session-based news recommendation with recurrent neural networks. *IEEE Access*, 7:169185–169203, 2019. doi: 10.1109/ACCESS.2019.2954957. 30, 75

[93] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 188–197, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1018. URL https://aclanthology.org/D19-1018. 13, 15, 79

[94] Lin Ning, Steve Chien, Shuang Song, Mei Chen, Yunqi Xue, and Devora Berlowitz. Eana: Reducing privacy risk on large-scale recommendation models. In *Proceedings of the 16th ACM Conference on Recommender Systems*, RecSys '22, page 399–407, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392785. doi: 10.1145/3523227.3546769. URL https://doi.org/10.1145/3523227.3546769. 3

[95] Xia Ning and George Karypis. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*, pages 497–506, 2011. doi: 10.1109/ICDM.2011.134. 29, 30

[96] Douglas W Oard, Jinmook Kim, et al. Implicit Feedback for Recommender Systems. In *Proceedings of the AAAI workshop on recommender systems*, volume 83, pages 81–83. Madison, WI, 1998. 4

[97] Zohreh Ovaisi, Shelby Heinecke, Jia Li, Yongfeng Zhang, Elena Zheleva, and Caiming Xiong. Rgrecsys: A toolkit for robustness evaluation of recommender systems. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, WSDM '22, page 1597–1600, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391320. doi: 10.1145/3488560.3502192. URL https://doi.org/10.1145/3488560.3502192. 32

[98] Umberto Panniello, Michele Gorgoglione, and Cosimo Palmisano. Comparing pre-filtering and post-filtering approach in a collaborative contextual recommender system: An application to e-commerce. In Tommaso Di Noia and Francesco Buccafurri, editors, *E-Commerce and Web Technologies*, pages 348–359, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-03964-5. doi: 10.1007/978-3-642-03964-5_32. 21, 63

[99] Roberto Pellegrini, Wenjie Zhao, and Iain Murray. Don't Recommend the Obvious: Estimate Probability Ratios. In *Proceedings of the 16th ACM Conference on Recommender Systems*, RecSys '22, page 188–197, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392785. doi: 10.1145/3523227.3546753. 91

[100] Roberto Pellegrini, Wenjie Zhao, and Iain Murray. Don't recommend the obvious: Estimate probability ratios. In *Proceedings of the 16th ACM Conference on Recommender Systems*, RecSys '22, page 188–197, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392785. doi: 10.1145/3523227.3546753. URL https://doi.org/10.1145/3523227.3546753. 3

[101] Ladislav Peska and Peter Vojtas. Off-line vs. on-line evaluation of recommender systems in small e-commerce. In *Proceedings of the 31st ACM Conference on Hypertext and Social Media*, HT '20, page 291–300, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370981. doi: 10.1145/3372923.3404781. URL https://doi.org/10.1145/3372923.3404781. 3

[102] Shameem A. Puthiya Parambath, Nicolas Usunier, and Yves Grandvalet. A coverage-based approach to recommendation diversity on similarity graph. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, page 15–22, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340359. doi: 10.1145/2959100.2959149. URL https://doi.org/10.1145/2959100.2959149. 66

[103] Abdulhakim A. Qahtan, Basma Alharbi, Suojin Wang, and Xiangliang Zhang. A pca-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, page 935–944, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336642. doi: 10.1145/2783258.2783359. URL https://doi.org/10.1145/2783258.2783359. 42

[104] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. Sequence-aware recommender systems. *ACM Computing Surveys*, 51(4), July 2018. ISSN 0360-0300. doi: 10.1145/3190616. URL https://doi.org/10.1145/3190616. 63

[105] Ahmed Rashed, Shereen Elsayed, and Lars Schmidt-Thieme. Context and Attribute-Aware Sequential Recommendation via Cross-Attention. In *Proceedings of the 16th ACM Conference on Recommender Systems*, RecSys '22, page 71–80, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392785. doi: 10.1145/3523227.3546777. URL https://doi.org/10.1145/3523227.3546777. 19

[106] T. Rehorek, O. Biza, R. Bartyzal, P. Kordik, I. Povalyev, and O. Podstavek. Comparing Offline and Online Evaluation Results of Recommender Systems. In *REVEAL '18: Proceedings of the Workshop on Offline Evaluation for Recommender Systems, 2018*, 2018. URL https://users.fit.cvut.cz/~rehorto2/files/comparing-offline-online.pdf. 91

[107] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, page 452–461, Arlington, Virginia, USA, 2009. AUAI Press. ISBN 9780974903958. doi: 10.5555/1795114.1795167. 23, 29, 31, 48

[108] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Recommender Systems Handbook*. Springer US, Boston, MA, 2011. ISBN 978-0-387-85820-3. doi: 10.1007/978-0-387-85820-3. URL https://doi.org/10.1007/978-0-387-85820-3. 32

[109] David Rohde, Stephen Bonner, Travis Dunlop, Flavian Vasile, and Alexandros Karatzoglou. RecoGym: A Reinforcement Learning Environment for the Problem of Product Recommendation in Online Advertising, 2018. URL https://arxiv.org/abs/1808.00720. 7

[110] Marco Rossetti, Fabio Stella, and Markus Zanker. Contrasting Offline and Online Results When Evaluating Recommendation Algorithms. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, page 31–34, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340359. doi: 10.1145/2959100.2959176. 91

[111] Yuta Saito, Aihara Shunsuke, Matsutani Megumi, and Narita Yusuke. Open Bandit Dataset and Pipeline: Towards Realistic and Reproducible Off-Policy Evaluation. *arXiv preprint arXiv:2008.07146*, 2020. doi: 10.48550/ARXIV.2008.07146. 7

[112] Arindam Sarkar, Dipankar Das, Vivek Sembium, and Prakash Mandayam Comar. Dual attentional higher order factorization machines. In *Proceedings of the 16th*

ACM Conference on Recommender Systems, RecSys '22, page 378–388, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392785. doi: 10.1145/3523227.3546789. URL https://doi.org/10.1145/3523227.3546789. 3

[113] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, page 285–295, New York, NY, USA, 2001. Association for Computing Machinery. ISBN 1581133480. doi: 10.1145/371920.372071. URL https://doi.org/10.1145/371920.372071. 30, 65, 80

[114] Teresa Scheidt and Joeran Beel. Time-dependent evaluation of recommender systems. In *Perspectives on the Evaluation of Recommender Systems Workshop (PERSPECTIVES 2021), September 25th, 2021, co-located with the 15th ACM Conference on Recommender Systems*, 2021. 63, 64

[115] Guy Shani and Asela Gunawardana. Evaluating recommendation systems. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 257–297. Springer US, Boston, MA, 2011. ISBN 978-0-387-85820-3. doi: 10.1007/978-0-387-85820-3_8. URL https://doi.org/10.1007/978-0-387-85820-3_8. 66, 80

[116] Ilya Shenbin, Anton Alekseev, Elena Tutubalina, Valentin Malykh, and Sergey I. Nikolenko. *RecVAE: A New Variational Autoencoder for Top-N Recommendations with Implicit Feedback*, page 528–536. Association for Computing Machinery, New York, NY, USA, 2020. ISBN 9781450368223. URL https://doi.org/10.1145/3336191.3371831. 29

[117] Pannaga Shivaswamy and Dario Garcia-Garcia. Adversary or friend? an adversarial approach to improving recommender systems. In *Proceedings of the 16th ACM Conference on Recommender Systems*, RecSys '22, page 369–377, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392785. doi: 10.1145/3523227.3546784. URL https://doi.org/10.1145/3523227.3546784. 3

[118] Ashudeep Singh and Thorsten Joachims. Fairness of Exposure in Rankings. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '18, page 2219–2228, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3220088. URL https://doi.org/10.1145/3219819.3220088. 32

[119] Padipat Sitkrongwong, Saranya Maneeroj, Pannawit Samatthiyadikun, and Atsuhiro Takasu. Bayesian Probabilistic Model for Context-Aware Recommendations. In *Proceedings of the 17th International Conference on Information Integration and Web-Based Applications amp; Services*, iiWAS '15, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450334914. doi: 10.1145/2837185.2837223. URL https://doi.org/10.1145/2837185.2837223. 75

[120] Paschalia (Lia) Spyridou, Constantinos Djouvas, and Dimitra Milioni. Modeling and validating a news recommender algorithm in a mainstream medium-sized news organization: An experimental approach. *Future Internet*, 14(10), 2022. ISSN 1999-5903. doi: 10.3390/fi14100284. URL https://www.mdpi.com/1999-5903/14/10/284. 3

[121] Harald Steck. Embarrassingly shallow autoencoders for sparse data. In *The World Wide Web Conference*, WWW '19, page 3251–3257, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366748. doi: 10.1145/3308558.3313710. URL https://doi.org/10.1145/3308558.3313710. 29, 30, 34, 65, 80

[122] Harald Steck. Markov random fields for collaborative filtering. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019. Curran Associates Inc. doi: 10.5555/3454287.3454778. 42, 51

[123] Zhu Sun, Di Yu, Hui Fang, Jie Yang, Xinghua Qu, Jie Zhang, and Cong Geng. *Are We Evaluating Rigorously? Benchmarking Recommendation for Reproducible Evaluation and Fair Comparison*, page 23–32. Association for Computing Machinery, New York, NY, USA, 2020. ISBN 9781450375832. URL https://doi.org/10.1145/3383313.3412489. 12, 13

[124] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, page 565–573, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355810. doi: 10.1145/3159652.3159656. URL https://doi.org/10.1145/3159652.3159656. 63

[125] Nam Khanh Tran, Andrea Ceroni, Nattiya Kanhabua, and Claudia Niederée. Time-Travel Translator: Automatically Contextualizing News Articles. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, page 247–250, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450334730. doi: 10.1145/2740908.2742841. URL https://doi.org/10.1145/2740908.2742841. 77

[126] Paula Cristina Vaz, Ricardo Ribeiro, and David Martins de Matos. Understanding the temporal dynamics of recommendations across different rating scales. In Shlomo Berkovsky, Eelco Herder, Pasquale Lops, and Olga C. Santos, editors, *Late-Breaking Results, Project Papers and Workshop Proceedings of the 21st Conference on User Modeling, Adaptation, and Personalization., Rome, Italy, June 10-14, 2013*, volume 997 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013. URL http://ceur-ws.org/Vol-997/umap2013_lbr_7.pdf. 31, 63, 76, 80

[127] Robin Verachtert, Lien Michiels, and Bart Goethals. Are We Forgetting Something? Correctly Evaluate a Recommender System With an Optimal Training Window. In Eva Zangerle, Christine Bauer, and Alain Said, editors, *Proceedings of the Perspectives on the Evaluation of Recommender Systems Workshop 2022*, volume 3228, Seattle, WA, USA, sep 2022. CEUR-WS.org. 21, 75, 79, 80, 81, 83, 90, 91

[128] Koen Verstrepen and Bart Goethals. Unifying nearest neighbors collaborative filtering. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, page 177–184, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450326681. doi: 10.1145/2645710.2645731. URL https://doi.org/10.1145/2645710.2645731. 29

[129] Koen Verstrepen, Kanishka Bhaduriy, Boris Cule, and Bart Goethals. Collaborative filtering for binary, positiveonly data. *SIGKDD Explorations Newsletter*, 19(1):1–21, sep 2017. ISSN 1931-0145. doi: 10.1145/3137597.3137599. URL https://doi.org/10.1145/3137597.3137599. 43

[130] João Vinagre, Alípio Mário Jorge, Marie Al-Ghossein, and Albert Bifet. *ORSUM - Workshop on Online Recommender Systems and User Modeling*, page 619–620. RecSys '20. Association for Computing Machinery, New York, NY, USA, 2020. ISBN 9781450375832. doi: 10.1145/3383313.3411531. URL https://doi.org/10.1145/3383313.3411531. 40

[131] João Vinagre and Alípio Mário Jorge. Forgetting mechanisms for scalable collaborative filtering. *Journal of the Brazilian Computer Society*, 18(4):271–282, 2012. doi: 10.1007/s13173-012-0077-3. 42, 63, 75, 76

[132] João Vinagre, Alípio Mário Jorge, and João Gama. Fast incremental matrix factorization for recommendation with positive-only feedback. In Vania Dimitrova, Tsvi Kuflik, David Chin, Francesco Ricci, Peter Dolog, and Geert-Jan Houben, editors, *User Modeling, Adaptation, and Personalization*, pages 459–470, Cham, 2014. Springer International Publishing. ISBN 978-3-319-08786-3. doi: 10.1007/978-3-319-08786-3_41. 40, 42

[133] Gerhard. Widmer and Miroslav. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996. doi: 10.1023/A:1018046501280. 42, 63

[134] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J. Smola, and How Jing. Recurrent recommender networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, page 495–503, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346757. doi: 10.1145/3018661.3018689. URL https://doi.org/10.1145/3018661.3018689. 30, 65, 80

[135] Chaolun Xia, Xiaohong Jiang, Sen Liu, Zhaobo Luo, and Zhang Yu. Dynamic item-based recommendation algorithm with time decay. In *2010 Sixth International Conference on Natural Computation*, volume 1, pages 242–247, 2010. doi: 10.1109/ICNC.2010.5582899. 31, 63, 74, 76, 77, 78, 86

[136] Shujian Yu and Zubin Abraham. Concept drift detection with hierarchical hypothesis testing. In *Proceedings of the 2017 SIAM International Conference on Data Mining (SDM)*, pages 768–776, 2017. doi: 10.1137/1.9781611974973.86. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611974973.86. 42

[137] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M. Jose, and Xiangnan He. A simple convolutional generative network for next item recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, WSDM '19, page 582–590, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359405. doi: 10.1145/3289600.3290975. 74

[138] Valentina Zanardi and Licia Capra. Dynamic updating of online recommender systems via feed-forward controllers. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '11, page 11–19, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450305754. doi: 10.1145/1988008.1988011. URL https://doi.org/10.1145/1988008.1988011. 43

[139] Eva Zangerle and Christine Bauer. Evaluating Recommender Systems: Survey and Framework. *ACM Computing Surveys*, 55(8), dec 2022. ISSN 0360-0300. doi: 10.1145/3556536. 6, 7, 95

[140] Yang Zhang, Fuli Feng, Chenxu Wang, Xiangnan He, Meng Wang, Yan Li, and Yongdong Zhang. How to retrain recommender system? a sequential meta-learning method. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 1479–1488, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380164. doi: 10.1145/3397271.3401167. URL https://doi.org/10.1145/3397271.3401167. 42

[141] Yin Zhang. GroRec: A Group-Centric Intelligent Recommender system integrating social, mobile and big data technologies. *IEEE Transactions on Services Computing*, 9(5):786–795, 2016. doi: 10.1109/TSC.2016.2592520. 75

[142] Wayne Xin Zhao, Junhua Chen, Pengfei Wang, Qi Gu, and Ji-Rong Wen. Revisiting alternative experimental settings for evaluating top-n item recommendation algorithms. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, page 2329–2332, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368599. doi: 10.1145/3340531.3412095. URL https://doi.org/10.1145/3340531.3412095. 12

[143] Ziwei Zhu, Yun He, Xing Zhao, and James Caverlee. Popularity Bias in Dynamic Recommendation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery amp; Data Mining*, KDD '21, page 2439–2449, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383325. doi: 10.1145/3447548.3467376. 90, 91

Summary

In this thesis, we describe solutions to problems that arise naturally when using recommender systems outside of the controlled environments common in academic research.

- In **Chapter 2**, we present how we set up controlled, offline experiments. In this, we make sure to avoid common pitfalls in evaluation, such as leakage and parameter optimisation on the test dataset. In this chapter, we use the library `RecPack`, as the framework for this evaluation, providing us with the building blocks to easily, quickly and faultlessly set up an experiment.
- In **Chapter 3**, we define and compare several methods to schedule model updates. Production recommender systems need frequent updating of the recommendation models to avoid staleness in the recommendations and to account for concept drift in the data stream. We compare smart schedulers based on the number of events, the unexpectedness of events, and the expected change in the model with the baseline time-based scheduler. We find that the smart schedulers are able to make a difference, allowing performance improvements, while not needing to increase the costs for training models.
- In **Chapter 4**, we look further at the impact of temporal dynamics on model quality. We show that the data used to train a recommendation model has a large impact on the quality of the model. By using only recent data points to train algorithms, we find that popularity becomes an almost unbeatable baseline in offline experiments. Further, we find that other, simple methods, such as ItemKNN benefit strongly from a reduction in training data, and can now outperform more advanced and expensive neural networks. These findings show that even in offline experiments, we need to take care to fully optimise our baselines in order to make fair comparisons.
- In **Chapter 5**, we continue the efforts from Chapter 4 to address the concept drift when training models. Except that we don't want to take the rough and indiscriminate method of ignoring most of the available data. Instead, we analyse all known methods for ItemKNN that include a time-decay factor and propose a framework that generalises all of these methods and introduces additional, untried combinations. We find that by applying the right decay, we can indeed improve on the crude forgetting mechanism proposed earlier.
- Finally, in **Chapter 6**, we look at the impact on recommendation quality and user engagement, when we show less popular items in a news context. We analyse the discrepancies between offline and online results and find that the popularity bias present in the data, also causes the evaluation to favour algorithms that recommend more popular items, whereas online results show that users are more likely to click on the recommendations when we punish popular items and show a more balanced list.

Samenvatting

In dit proefschrift beschrijven we oplossingen voor problemen die op natuurlijke wijze ontstaan bij het gebruik van recommender systemen buiten de gecontroleerde omgevingen die gebruikelijk zijn in academisch onderzoek.

- In **hoofdstuk 2** presenteren we hoe we gecontroleerde, offline experimenten opzetten. Hierbij zorgen we ervoor dat veelvoorkomende valkuilen bij evaluatie, zoals lekkage en parameter optimalisatie op de test dataset, worden vermeden. In dit hoofdstuk gebruiken we de library `RecPack`, die ons de bouwstenen verschaft om eenvoudig, snel en foutloos een experiment op te zetten.
- In **hoofdstuk 3** definiëren en vergelijken we verschillende methoden voor het plannen van model updates. Productie recommender systemen moeten regelmatig de aanbevelingsmodellen bijwerken, om te voorkomen dat de aanbevelingen star worden en om rekening te houden met concept drift in de datastroom. Wij vergelijken slimme planners op basis van het aantal gebeurtenissen, de onverwachtheid van gebeurtenissen en de verwachte verandering in het model met de basisplanner op basis van tijd. Wij vinden dat de slimme schedulers een verschil kunnen maken en prestatieverbeteringen mogelijk maken, terwijl de kosten voor het trainen van modellen niet verhoogd hoeven te worden.
- In **hoofdstuk 4** kijken we verder naar de invloed van tijdsdynamiek op de kwaliteit van het model. We laten zien dat de gegevens die gebruikt worden om een aanbevelingsmodel te trainen een grote invloed hebben op de kwaliteit van het model. Door alleen recente datapunten te gebruiken om algoritmen te trainen, vinden we dat populariteit een bijna onverslaanbare baseline wordt in offline experimenten. Verder stellen wij vast dat andere, eenvoudige methoden, zoals ItemKNN, sterk profiteren van een vermindering van het aantal trainingsgegevens, en nu beter kunnen presteren dan meer geavanceerde en dure neurale netwerken. Deze bevindingen tonen aan dat wij zelfs in offline experimenten onze basislijnen volledig moeten optimaliseren om eerlijke vergelijkingen te kunnen maken.
- In **hoofdstuk 5** zetten we de inspanningen uit hoofdstuk 4 voort om conceptdrift bij het trainen van modellen aan te pakken. Alleen willen we niet de ruwe en ongedifferentieerde methode toepassen om de meeste beschikbare gegevens te negeren. In plaats daarvan analyseren wij alle bekende methoden voor ItemKNN die een time-decay factor bevatten en stellen wij een kader voor dat al deze methoden generaliseert en aanvullende, niet beproefde combinaties introduceert. Wij vinden dat wij door de juiste decay toe te passen, het eerder voorgestelde ruwe vergeetmechanisme inderdaad kunnen verbeteren.
- Tot slot kijken we in **hoofdstuk 6** naar het effect op de aanbevelingskwaliteit en de betrokkenheid van gebruikers wanneer we minder populaire items tonen in een

nieuwscontext. We analyseren de discrepanties tussen offline en online resultaten en stellen vast dat de in de gegevens aanwezige populariteitsbias er ook toe leidt dat de evaluatie algoritmen bevoordeelt die populairdere items aanbevelen, terwijl online resultaten aantonen dat gebruikers eerder geneigd zijn op de aanbevelingen te klikken wanneer we populaire items bestraffen en een evenwichtiger lijst tonen.