Addressing the Next-Poster Problem: A Hybrid Recommender System for Streaming Platforms

Snorre Alvsvåg

Supervisor: **Full. Prof. Dr. Christoph Trattner**
Co-Supervisor: **Assoc. Prof. Dr. Alain Starke**

Industry aff.: **Dr. Lars Skjerven and Astrid Tessem**

Thesis for Master Degree at the University of Bergen, Norway
Department of Information Science and Media Studies

2024

Year:	2024
Title:	Addressing the Next-Poster Problem: A Hybrid Recommender System for Streaming Platforms
Author:	Snorre Alvsvåg
Github Repository:	https://github.com/sfimediafutures/MA_Snorre-Alvsvaag

Acknowledgements

I would like to thank my main supervisor, Full. Prof. Dr. Christoph Trattner and my co-supervisor Assoc. Prof. Dr. Alain Starke for their combined effort and guidance through this thesis. Their feedback was invaluable. Further I would like to thank my industry co-supervisors Dr. Lars Skjerven and Astrid Tessem as well as the extended Content Discovery Team at TV 2 Norway for their guidance and for allowing me to play around with their software, helping me in understanding the complicated nature of deployed recommender systems. I would also like to thank Dr. Dietmar Jannach for his input on the first rounds of results, allowing me to understand the implications therein. Finally, I would like to extend my thanks to the research centre SFI MediaFutures, where I grew my appetite for research alongside some of the most talented individuals and collective I know. Lastly I would like to thank TV 2 Norway for allowing me to use their icons for the visualizations, whilst rudimentary, it made the process more enjoyable for me, and hopefully also for you, the reader.

This work was supported by industry partners and the Research Council of Norway with funding to MediaFutures: Research Centre for Responsible Media Technology and Innovation, through the Centres for Research-based Innovation scheme, project number 309339.

Abstract

Recommendation strategies in the Movie domain is varied, but has been shown to aid users in finding content they like. On video streaming-platforms such as TV 2 Play, the user is exposed to several different arenas where they can find something they would like to watch, be that the landing-page of the streaming site, or a suggestion for something more to watch after they concluded a movie or series. These are all areas where Recommender strategies can recommend something based on either the preferences of the user, or in the case of the concluded movie or series, something more to watch based on *what they just watched*. This latter aspects, being what I refer to as the *next-poster problem* in this thesis, is not a largely explored area of research, where previous actors have simply utilized the already established Collaborative Filtering (CF) model concerned with the user's preferences without considering *what the user just watched*. Here I show that a solution to the next-poster problem is to combine the CF model with a Sequence Aware approach based on Markov Chains, finding an increase in implied user satisfaction over the baseline CF approach. Through an online evaluation on the streaming platform TV 2 Play, I show that using a Hybrid approach to solve the next-poster problem rather than a traditional CF model leads to a lessening in user engagement such as CTR, but an increase in the clicks resulting in a user actually watching the content, this being our implied user satisfaction. Further as a result of this online evaluation, I am able to show that its possible to find the best configuration for a Hybrid model based on Sequence Aware and CF approaches deployed in a real life scenario, through offline evaluation. The results allows me to showcase the importance of considering Sequence of items when recommending for the next-poster problem, and to show that an offline evaluation can imply results in a real world scenario, when considering the Movie domain. Although an improvement, this thesis also shows that there are many more avenues to consider for the next-poster problem.

Contents

Chapter 1

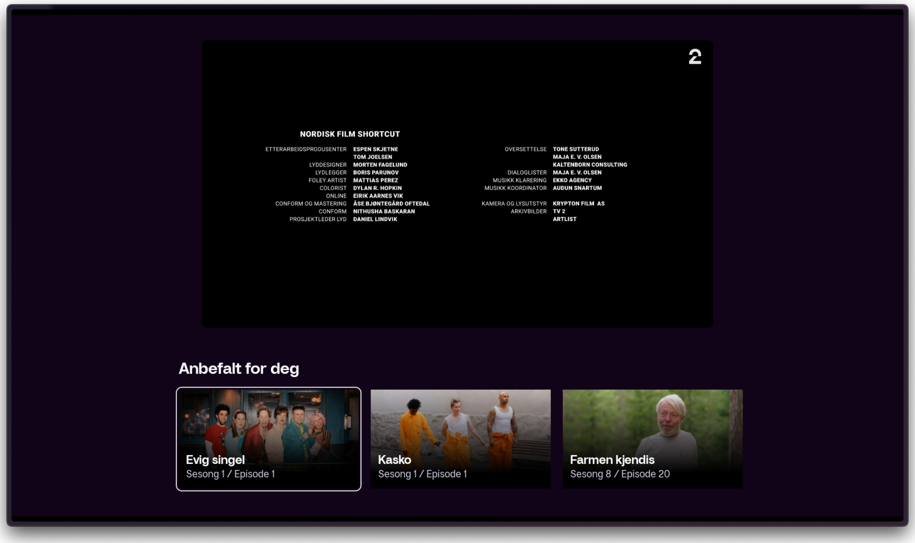
Introduction

As the internet grew in reach and bytes, it did not take long before it outgrew man's ability to navigate, read, and sort the wealth of information available. In this new era of information, research fields such as Recommender Systems gained traction, due to its ability to sort and filter out this information based on relevancy for the individual user. Recommender Systems can help with finding what is right for you and you alone and have been introduced to aspects of modern life, such as E-commerce sites, Music apps, Short-form video apps like TikTok, News, Food, the list goes on. One of these areas, or Domains as they are referred to, is the Movie Domain, which is also the focus of this thesis. In particular, the rise of streaming services has revolutionized the way we consume media, but also introduced a new set of problems related to how we consume, one of these is the issue I introduce in this thesis known as the *Next-Poster Problem*.

1.1 Motivation

Streaming services have been around for a while now, the pioneer Netflix[1] laid the grounds for what a streaming service should be, and reshaped how we as a society consumed media. Following the rise of streaming platforms, traditional linear TV fell in popularity (SSB, 2024), and in its stead, we started consuming content over the internet, on one streaming platform or the other. Today there are a plethora of streaming platforms available to the public, and fierce competition to keep the user on your specific platform. Discussing the length of how streaming platforms keep the users engaged across the service is out of scope for this thesis, but one tool which has been around for a while is the play-next aspect of TV series. When a user is done watching an episode of a show, we start playing the next episode automatically, this keeps the user engaged without requiring them to take action. This aspect is universally applied across the streaming services available domestically in Norway, and have been for a while. But what happens when there are no more episodes to show? Or if the user has watched a movie? The user sat down planning on watching two or three episodes of a show, but suddenly the next episode isn't airing until next week, and our

[1] https://www.netflix.com



NORDISK FILM SHORTCUT

ETTERARBEIDSPRODUSENTER	ESPEN SKJETNE
	TOM JOELSEN
LYDDESIGNER	MORTEN FAGELUND
LYDLEGGER	BORIS PARUNOV
FOLEY ARTIST	MATTIAS PEREZ
COLORIST	DYLAN R. HOPKIN
ONLINE	EIRIK AARNES VIK
CONFORM OG MASTERING	ÅSE BJØNTEGÅRD OFTEDAL
CONFORM	NITHUSHA BASKARAN
PROSJEKTLEDER LYD	DANIEL LINDVIK

OVERSETTELSE	TONE SUTTERUD
	MAJA E. V. OLSEN
	KALTENBORN CONSULTING
DIALOGLISTER	MAJA E. V. OLSEN
MUSIKK KLARERING	EKKO AGENCY
MUSIKK KOORDINATOR	AUDUN SNARTUM
KAMERA OG LYSUTSTYR	KRYPTON FILM AS
ARKIVBILDER	TV 2
	ARTLIST

Figure 1.1: The next poster at TV 2 play, at mid center we see the post credits of whatever show or movie the user just watched, and below three recommendations that are "Recommended for you", in this case being three different television shows.

user is left in a position where they need to make a decision of *what to watch next*. This is a key moment where a platform can and should introduce the user with something to watch. Whilst conducting a broad exploration of what streaming services presented to the user in these scenarios, I found that almost all streaming services in Norway (see Table 1.1) had some kind of *next-poster*, see Figure 1.1, that they presented to the user, however, the approaches differed somewhat depending on the platform. Note that this exploration was not meant to be a survey of the different approaches, but was done to feel out the market. Querying the services on their desktop services (e.g. browser), mobile apps (IOS) and TV (Android TV), I found that all with the exception of HBO Max had some sort of next-poster. The most popular approach was to present the user with something else, in the case of Netflix a promo for the latest Netflix Special played, and finally for Discovery+, whilst having a visually lacking end-poster, automatically started playing a new episode of a different show. To the best of our knowledge, these approaches are either editor-controlled, e.g. that a human is in the loop choosing the best alternative for a given show or movie, or that it is based on a recommendation approach. Whilst an editor can choose what they believe is best given the current content by making an expert opinion, there is simply too much content on any given streaming platform for a human to cover all of these. Further, applying such a catch-all solution does not account for the users' preferences. To mitigate this, platforms such as the Norwegian streaming platform TV 2 Play employ a Collaborative Filtering (CF) model, a recommendation technique often used to recommend content to a user based on their preferences. The downside to such an approach comes from the context our user finds themselves in, as mentioned our user is trying to find something *to*

Table 1.1: Paid Streaming Services in Norway and their respective market shares, from Fam Vivian Bekkengen (2024) (Norwegian).

Streaming Service	Market Share (%)	Has Next-poster
Netflix	69	Yes
TV 2 Play	40	Yes
HBO Max	38	No
Viaplay	37	Yes
Disney+	36	Yes
Discovery+	19	Yes
Prime Video	11	Yes
Apple TV+	7	Yes

watch next, not just something to watch. A branch of recommendation strategies known as Sequence Aware Recommendation systems are often tasked with the same problem, known as trying to *adapt the context* of our user. I am interested in seeing if applying such a *context adaptation* approach would increase the user's engagement with the provided items.

To the best of my knowledge none of the media houses apply this context adaptation aspect in their next-poster's, nor could I find any academic literature directly considering the next-poster as a medium for recommendation, regardless of recommender strategies. Further, previous Sequence Aware approaches have been explored primarily in relation to Music, E-commerce, or other areas where the *content* is relatively short, and where the *events* that are found surrounding this content such as adding something to your online shopping cart, skipping a song, or spending time on a certain part of a website, are weighted when making a recommendation to a user. When dealing with movies or series the content itself is much longer, and whilst it's hard to gauge a user's interest based on e.g. ten songs. There's an argument that you can gauge a user's interest based on ten Movies, or even more, ten complete Series, due to how the time our user invested in those ten Movies or Series is much greater than the time our user invested in those ten songs. With this in mind, rather than exploring the effect of a purely Sequence Aware approach, considering the sequence of events of a user in isolation, I intend to employ a so-called Hybrid approach, where I combine the user's preferences given any Movie or Series gauged from the Collaborative Filtering approach, together with the *adapted context* of our user, ergo what our user is trying to achieve, through a Sequence Aware approach. Finally, as there is very little research on these Sequence Aware approaches in terms of movies and series, understanding exactly which approach would work best given the problem statement requires a wide exploration. To test them all can be costly in a live production-grade system, seeing as implementing models in a live environment is not easy as I will detail in this thesis, and I therefore also wish to explore if I am able to emulate the performance of our approach through traditional simulation techniques found in literature, known as offline evaluation.

1.2 Problem Formulation

This thesis addresses what I refer to as the 'next-poster' problem by attempting to combine the user's preferences through a Collaborative Filtering model, with the adapted context of our user through a Sequence Aware approach. The primary concerns of this investigation are: how does the Hybrid model perform in comparison to the traditional CF model found on the streaming platform TV 2 Play when tasked with the next-poster problem, and is it possible to simulate the performance of this Hybrid model when conducting a controlled offline simulation? I will be using the metrics Mean Reciprocal Rank (MRR) and Click Through Rate (CTR) as user engagement metrics, both indicating if a user clicked a provided recommendation or not, the former considering the order these items are presented in, and the latter simply considering if it was clicked at all. Alongside this, in our live experiment I will be adding some metrics indicating if our users consumed the content presented to them, or if they simply clicked and subsequently exited, in other words, the success of the click. With this in mind, I can formulate two research questions:

- **RQ1**: To what extent does a Hybrid model outperform the traditional Collaborative Filtering model in regards to user engagement and click success?
- **RQ2**: To what extent can the outcome of an online evaluation be predicted based on an offline evaluation when utilizing a Sequence Aware approach in the Movie domain?

1.3 Objectives

The primary objective of this thesis is to develop and evaluate a Hybrid recommendation system that integrates Collaborative Filtering with a Sequence Aware Recommendation strategy, specifically tailored for the *next-poster* scenario in streaming services. This objective is broken down into several goals:

- **To design a hybrid model** that combines the broad user preference insights of collaborative filtering with the contextual relevance of Sequence Aware recommendation techniques. This model aims to improve the relevancy and accuracy of recommendations made in the next-poster scenario.
- **To implement and test the model in a real-world environment** through the collaboration with TV 2 Play. This includes setting up the system within the existing infrastructure, ensuring seamless integration and real-time recommendation capabilities.
- **To utilize offline evaluation techniques and insights** from Sequence Aware models to optimize and evaluate the performance of the hybrid model, ensuring its readiness for subsequent online evaluation.
- **To analyze user engagement and click success** with the recommendations provided by the hybrid model. This involves tracking key metrics such as click-through rates

and watch time within TV 2 Play, to gauge the impact of improved recommendations on user engagement.

- **To enhance the academic and industry understanding** of Sequence Aware recommendation systems in streaming platforms, with a focus on the differences between online and offline evaluation methods. This research aims to document and analyze these distinctions, providing insights that will inform the development of more effective recommendation strategies for both academia and industry.

These objectives are designed to ensure that the research not only advances theoretical knowledge but also applies these insights in practical, impactful ways in the streaming industry.

1.4 Contribution

This thesis presents the design, evaluation, and industry implementation of a Hybrid recommender system that integrates Collaborative Filtering with a Sequence Aware approach. The Hybrid model addresses the "next-poster" problem in streaming services by combining user preferences via Collaborative Filtering with the contextual adaptation provided by Sequence Aware methods. The key contributions of this thesis are as follows:

- **Novel Evaluation in the Movie Domain:** To the best of my knowledge, this is the first time a Sequence Aware approach, incorporated within a Hybrid model, has been evaluated online in the Movie domain.

- **Offline Evaluation Insights:** This thesis shows that offline evaluations can effectively indicate which configurations of a Hybrid recommender model will perform best in real-life scenarios in regard to the Movie domain.

- **Industry Application:** By applying the Hybrid model within a real-world streaming service, this thesis enhances the adds to the knowledge of industry applications for recommender systems.

1.5 Outline

This thesis is divided into six sections, firstly I introduce the concepts and background for this thesis in Section 2, then the methodology in Section 3. Following are the results in Section 4, then the discussion in Section 5. I then conclude the thesis with Sections 6, where I discuss the Limitations and finally conclude the thesis.

Chapter 2

Background

This Section provides an overview of the fundamental concepts and methodologies that underpin recommender systems. It begins with a discussion on Collaborative Filtering, a popular method used in many recommendation algorithms. Following this, the specifics of Sequential Recommender Systems are discussed, which consider the order of user interactions to provide more contextually relevant suggestions. The integration of these approaches into Hybrid Recommender Systems is then examined, highlighting their combined strengths. Additionally, the unique challenges and characteristics of the movie domain are discussed, providing context for the specific application of these techniques. Finally, the architectural implementation of recommender systems and related works are reviewed.

2.1 Recommender systems

Recommender systems are at the core mathematical algorithms that predict and suggest items or content to users based on their individual preferences and behaviours (Jannach et al., 2010). By analyzing data such as past interactions, ratings, and contextual information, these systems provide personalized recommendations that are relevant to the user. They are widely used in areas like e-commerce, streaming services, social media, and online advertising to help users discover content they might otherwise miss. The primary goal of recommender systems is to enhance user experience by delivering tailored suggestions, thereby improving user satisfaction, engagement, and retention. Due to the range of applications in which recommender systems have been applied, the methods used vary greatly (Roy & Dutta, 2022). This variance is often related to the domain we are trying to solve for, the data that is available, or the objective of the recommender system. Generally, recommender systems are broadly categorized into three categories, Collaborative Filtering, Content-Based, and Knowledge-Based recommender systems (Jannach et al., 2010). Out of the three, Collaborative Filtering is the most popular method (Roy & Dutta, 2022), and is also part of the focus of this thesis.

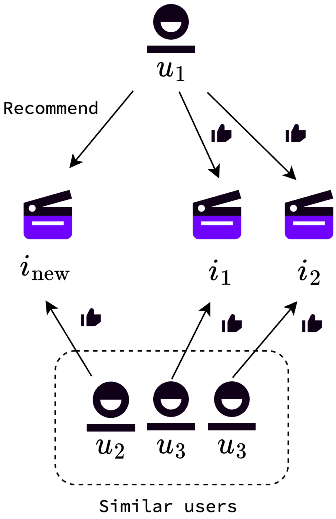


Figure 2.1: User-based Collaborative Filtering.

2.1.1 Collaborative Filtering

Collaborative filtering (CF) is not a new method and has been around for close to three decades. CF is tasked with providing a personalized recommendation based on patterns of usage from one user to another (Sarwar et al., 2001; Jannach et al., 2010). To give an example in the context of this thesis's medium, TV 2 Play, we observe a user's pattern on an item as u_i. u is the user Ken, when Ken watches a movie, the ID of that movie i alongside a usage pattern, such as how long Ken watched this movie. This information will, over time, result in a profile P_u representing Ken's watch pattern. Through this profile, it's possible to find similar profiles to Ken by comparing his profile to other users utilizing varying methods. Finally, by looking at the movies that these similar users liked, the CF system assumes that Ken would like these movies as well. This approach is known as User-based Collaborative Filtering, where the assumption is that if one user likes an item, a user sharing that user's interest would also like that item, (see Figure 2.1).

In 2001 Sarwar et al. (2001) proposed another method called Item-based collaborative filtering, which addressed the known issues of sparsity and scalability found in User-based CF. Item-based CF takes a different approach where it takes the user's pattern on an item, and rather than looking at other users, it looks at the similarities between items, and recommends an item i to a user u if it's similar to the items that our user has previously rated high, see Figure 2.2.

Both User-based and Item-based CF, are part of the branch of CF models known as *Memory-Based Collaborative Filtering* (Jannach et al., 2010; Roy & Dutta, 2022). The User-based and Item-based models both rely on the rating database to be held in memory for inference[1], hence the categorization of memory-based. Another branch of CF models

[1] Inference is used broadly to describe how we interact with the model, e.g. how we recommend for a user at

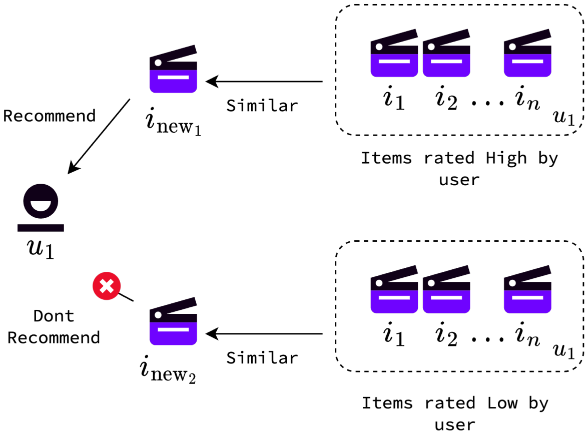


Figure 2.2: Item-based Collaborative Filtering.

are known as *Model-Based Collaborative Filtering* (Jannach et al., 2010). Model-Based CF is often recognized by the pre-computation or pre-processing of data, where the system inference with a model *built on the data*, rather than inferencing with the raw rating data itself. Although memory-based techniques are theoretically more precise than model-based techniques, memory-based approaches are limited by factors such as scalability (Jannach et al., 2010). Consider a bookshop in the centre of London. If all reoccurring customers were counted, the number would most likely not be on the scale of millions, and the selection of available items would be limited by the physical space of the store. The store owners wish to recommend books to customers based on their purchases to drive more sales. Implementing a memory-based CF model for this purpose would not significantly impact the memory of any modern computer, due to the limited physical space of the store (items) and the number of customers able to visit the store (users). However, if this bookshop were Amazon.com, with tens of millions of users and an unrestricted number of items, implementing a memory-based CF model would be more challenging. Running such a model might not be economically beneficial due to the high memory consumption. In this thesis, I utilize two model-based Collaborative Filtering approaches, Matrix Factorization (Section 2.1.3) and a Markov Chain implementation (Section 2.2.2).

2.1.2 Implicit and Explicit feedback

Any model is ultimately dependent on the data provided to it. In the earlier example, Ken's usage pattern was described, this type of user pattern would in literature be known as *implicit feedback* and can be described as indirectly inferring user opinion through observation (Oard et al., 1998; Koren et al., 2022). This task of inferring users' opinions about items through observation is one of the core challenges in recommendation systems (Roy & Dutta, 2022), and is vital to the performance of the system. Examples of *implicit*

run-time by providing an item-id or user-id to the model.

feedback in relation to our context, can be how long a user watched a movie, which movies the user has clicked on the homepage (Click Through Rate), and search terms from the user. Although *implicit feedback* has the ability to be continuously gathered, without disrupting our users, it is an assumption of user interest, and not a true measurement of interest. *Explicit feedback* is, as the name entails, an explicit measurement of user interest provided by the user. These explicit measurements are inherently a more precise way of gathering a users opinion and are often provided on a Likert scale from 1-5 or 1-7 where the range is from "Stronly dislike" (1) and "Strongly like" (5/7) (Jannach et al., 2010; Koren et al., 2022). Again in relation to our context, *explicit feedback* would be a user's rating of a movie or TV show on some scale, be that a 5-star scale, or a 10-point scale. Another example is the binary "Like" or "Dislike" which is present on the TV 2 Play platform.

Although *explicit feedback* is considered as a more precise way to infer user opinion, it has issues such as data scarcity, where one user often only rate a few items (Jannach et al., 2010), user-impairment, where asking a user for feedback might seem as intrusive, and cold-start issues, where new items with no ratings are left out of the recommendation (Jannach et al., 2010). *Implicit data* is also often more readily available, as big data trends have led to opportunistic data storage, capturing user information or item information at large, providing an ample opportunity to utilize this data to enhance a business's product through e.g. recommendations. In the Movie Domain, using implicit data is marginally more popular than using explicit data (Véras et al., 2015; Roy & Dutta, 2022), particularly in the context of a technique known as Matrix Factorization. In 2008 Hu et al. (2008) introduced a model for their case of a TV-Shows Recommendation Engine, proving how using such a technique showed impressive results.

2.1.3 Matrix Factorization

Latent factor models quickly became state of the art in the Movie and TV domain due to its well-performing nature and scalability (Véras et al., 2015; Koren et al., 2022). The goal of any latent factor model is to uncover latent (hidden) factors from the data which represents users or items as a lower dimension vector (Jannach et al., 2010). These factors can then be used to calculate a score $\hat{r}_{u,i}$[2] for a user u on an item i. Matrix factorization by singular value decomposition are one of these models, and proved to be the basis for many of the top-performing algorithms in the late 2000s in regards to the movie domain (Hu et al., 2008; Véras et al., 2015; Jannach et al., 2010). Matrix factorization maps users and items into a joint latent factor space of dimensionality f, so that user-item interactions are modeled as inner products of that space (Koren et al., 2009). The dimensionality of f is what I refer to when discussing *factors* in the models, this number is oftentimes a value between 20-100 (Koren et al., 2022), and varies on the implementation. Too many factors might lead to sparse representations which do not represent the latent pattern in the data, and too few factors can lead to over-fitting.

A basic implementation of matrix factorization is well presented in Koren et al. (2009), and can be described as such: Each item i is associated with a vector $q_i \in \mathbf{R}^f$, and each user is associated with a vector $p_u \in \mathbf{R}^f$, $\mathbf{R}^f$ is the joint latent matrix mentioned earlier, of

[2] Note that we formulate the predicted score for r as $\hat{r}$

dimensionality f. For any item i, the factor-vector q_i measure the extent to which the item posses those factors, positive or negative. For any user u, the factor-vector p_u measure the extent of interest the user has in items that are high on the corresponding factors, be that negative or positive. By computing the dot product, $q_i^T p_u$, the interest of user u in item i can be approximated, denoted as $r_{u,i}$. Therefore, in this scenario, the scoring function $\hat{r}_{u,i}$ can be defined as:

$$\hat{r}_{u,i} = q_i^T p_u \tag{2.1}$$

The dot product between two vectors $x, y \in \mathbf{R}^f$ is defined in Koren & Bell (2011) as:

$$x^T y = \sum_{k=1}^{f} x_k \times y_k \tag{2.2}$$

Computing the factors q_i and p_u in a way so that its possible to find and represent the latent patterns in our data is one of the major challenges when working with matrix factorization (Koren et al., 2009; Jannach et al., 2010). The goal is to be able to find representations that can help us connect users' interests to items, sometimes these representations are obvious, such as latent items genre patterns, but they can also be impossible to understand (Jannach et al., 2010) To bring back the user Ken, where traditional neighbor-based recommender systems are more akin to what was described previously, the focus would be on direct comparisons of user-item interactions. For example, if Ken liked movie A and B, and Kate liked B and C, then it would be inferred that Ken should like movie C. For matrix factorization, the focus shifts to how much Ken likes the latent factors produced by the model. For example, a latent factor could be movies that contain cars[3]. If Kate shares this interest, items that score well on this factor can be recommended to her, assuming she and Ken have other similar interests. The strength of matrix factorization lies in its ability to identify latent factors, such as movies with cars, which might be difficult for humans to find and categorize. The model can identify these patterns and leverage them for recommendation strategies.

As entailed by Koren et al. (2022, 2009); Hu et al. (2008); Jannach et al. (2010) there have been several different techniques employed to compute latent factors, these include *probabilistic Latent Semantic Analysis* (pLSA) (Hofmann, 2004), neural networks (Salakhutdinov et al., 2007), *Latent Dirichlet Allocation* (LDA) (Blei et al., 2003), *Principal Component Analysis* (PCA) (Goldberg et al., 2001), and models that are induced by factorization of the user-item matrix (Koren et al., 2022). This latter method, being the aforementioned matrix factorization by Singular Value Decomposition (SVD) gained great popularity thanks to its accuracy and scalability (Hu et al., 2008; Koren et al., 2022). SVD originates from the information retrieval field but was utilized alongside PCA in the early 2000's to identify latent factors by factorizing the user-item matrix (Koren et al., 2022). In contrast to document matrices in information retrieval, the user-item matrices are often very sparse, as users only rate a handful of items, and due to how Conventional SVD is undefined[4] when knowledge about the matrix is incomplete. To combat this imputation[5]

[3] Finding such a niche factor is one of the strengths of matrix factorization. Although it may sound unusual, this could be a factor if enough users provide a clear pattern for movies containing cars.

[4] SVD involves operations such as matrix multiplications and eigenvalues, something which requires the availability of a complete dataset (Aggarwal et al., 2016)

[5] Imputation is the process of replacing missing data with substituted values to complete the dataset though e.g.

was used in early research (Koren et al., 2009), but more recent works applied on explicit dataset suggested only using observed ratings, mitigating the resulting over-fitting with a regularization strategy (Aggarwal et al., 2016; Koren et al., 2022). An example of such a regularized model is presented by Hu et al. (2008) as:

$$\min_{q_*,p_*} \sum_{(u,i)\in\mathbf{K}} (r_{u,i} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2) \tag{2.3}$$

Equation 2.3 shows the formula for a regularized squared error on the set of known ratings, **K**. This formula is used to learn our factor vectors p_u and q_i by minimizing the error on every p and q for each known rating pair $(u,i) \in \mathbf{K}$,

$$\lambda(\|q_i\|^2 + \|p_u\|^2) \tag{2.4}$$

is the regularizing part of the model, where the parameter λ is used to regularize the magnitude of observations. L2 Normalization is applied to the feature vectors to penalize large values, reducing over-fitting. For a vector v the L2 norm squared is defined as $||v||^2 = v_1^2 + v_2^2 + ...v_n^2$, λ is a fixed value and is often found through cross-validation (Koren et al., 2009).

Scalability has so far been credited as one of the main advantages of matrix factorization. Previously, a scoring matrix with the shape of (u,i) was used, but through matrix factorization (or latent factor models in general), a much more compact matrix is obtained, resulting in a matrix (q_i, p_u) with size (f, f). However, solving for these factors in Equation 2.3 remains a computational burden. To help ease this burden, a few methods have been introduced, the most popular of which are Stochastic Gradient Descent (SGD) and Alternating Least Squares (ALS) (Koren et al., 2009).

SDG became popular due to its ease of implementation and speed, parsing each given training case, computing the associated prediction error, and finally modifying the parameters q_i and p_u by a set value of γ in the opposite direction of the gradient[6]. ALS, on the other hand, showed promise due to two main factors (Koren et al., 2009), firstly it's well optimized for large datasets, as rather than solving for every q_i and p_u at the same time, we keep every $q_i \in \mathbf{R}^f$ fixed, solving only every $p_u \in \mathbf{R}^f$, then fixing p_u and solving for q_i. The process is repeated n times [7] and allows for the computation to scale linearly with the amount of data. As explained in detail by Aggarwal et al. (2016), this method also allows for parallelization, which results in impressive computation time. The second factor has to do with implicit and explicit data. Given an explicit dataset, using SDG would still be considered favourable, however when dealing with implicit datasets, the training set cannot be considered sparse (Hu et al., 2008), and therefore the process of looping over the entire training dataset would not be practical.

ALS and other matrix factorization methods have been applied in different recommender system domains for some time now, and through the popular python library Impicit [8], they are readily available for commercial and academic use.

averaging (Little & Rubin, 2019)

[6] For a deeper understanding of this, see Aggarwal et al. (2016) ch. 3.6.4.1

[7] This repetition is what we refer to when discussing the 'iteration' variable in the ALS application.

[8] https://github.com/benfred/implicit

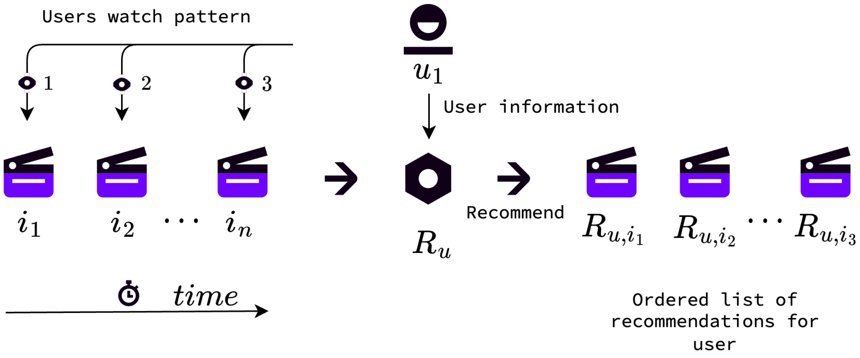


Figure 2.3: A basic visualization of a Sequence Aware recommender system.

2.2 Sequential Recommender systems

Sequential Recommender system is tasked with recommending items to a user based on the sequential patterns that those users produce, or the sequential patterns of which those items occur together. A sequential pattern is a time-sensitive array of events, e.g. indicating actions or reactions from a user, although a sequential pattern could also be a set of events related to an item, e.g. the football first hit the foot of a player, then the goal[9]. This connection between event and time is often what characterizes sequential patterns, in other words if the temporal aspect is ignore removed, there might not be any order to the data. In many scenarios it only makes sense to recommend an item in the context of another, e.g. it does not make sense to recommend a user a lens for their Sony camera, if the user has a Canon camera in their online shopping cart. Although this example could be argued in the context of knowledge-based recommenders as well, the sequence of the user putting said Canon camera in their shopping cart, and then receiving the appropriate recommended Lens, is the essence of the sequential aspect. The example above describes a sequence of actions. Expanding on such an example in the Domain of our thesis, a sequence could be: user Ken first watched an episode of a show, then a movie, and then another episode of a different show. All of these sequences produced by different users are aggregated, and then recommendations are made for any other user based on their produced sequences or their first action. For example, if Ken started watching a new movie i, based on the mined sequences from other users who watched i, it can be determined that movie j should be recommended. A basic visualization of a Sequence Aware recommender can be seen in Figure 2.3.

Whereas a Collaborative Filtering or a more traditional recommendation approach tries to predict the score $\hat{r}$ for user u on any item $i \in I$, a Sequence Aware model attempts to find the best fitting permutation[10], π at length k of the corpus of available items I that maximises the score for the user. Quadrana et al. (2018) formalises this problem with the following

[9] An example would be: $(\{[hit, 19:41:10], [goal, 19:41:11]\})$

[10] Permutation (denoted π) is used to describe an ordering of items for any given set, e.g. $\{i_1, i_3, i_2\}$ is a permutation of the set $\{i_1, i_2, i_3\}$

formula[11]:

$$\forall u \in U, l'_u = \arg\max_{l \in L^*} f(u,l). \tag{2.5}$$

Where U is the set of all users u, f is the function that produces a utility score for a given sequence L for any user u, therefore $f : U \times L^* \rightarrow R$. L^* is the set of possible permutations L up to the length of k of the powerset[12] of I, and finally l'_u is the element of L that maximizes the score for any user u.

Finding a function f that produces the best score for the users completely depends on the problem the recommender is trying to solve, and the data it is provided. As is argued in Quadrana et al. (2018), the function, or rather model, could be tasked with trying to guess the next hidden item to show to the user, such an issue is common and also what I described previously in regards to the Collaborative Filtering approaches, however, Sequence Aware models are also tasked with other aspects such as *Context adaptation*, *Trend detection*, *Repeated Recommendations* and *Consideration of Order Constraints and Sequential patterns*. Out of these four, the most interesting aspect in regards to this thesis is *Context Adaptation*, but for further reading on these, I recommend Quadrana et al. (2018).

2.2.1 Context Adaptation

Utilizing the context of the user to perform recommendations is nothing new (Adomavicius et al., 2022), considering that applying context to a recommendation model improves the recommendation greatly (Adomavicius & Tuzhilin, 2011). Context however is often considered as something physical, such as the medium a user is consuming content in, e.g. the TV, Mobile phone, or Laptop. It can also be context information which is not directly observable, such as the user's watching intent. This latter form of context is known as *interactional context* (Quadrana et al., 2018), which are often derived from the user's most recent actions, and behavioural patterns. This context is often important in systems operating with anonymous users, e.g. the system lacks any user-profiles to base the recommendation on. The question is then, based on the interactional context found for the user, how can the user's situation and goals be understood to perform a recommendation thereafter? Depending on the data available, three approaches can be distinguished:

- *Last-N Interaction based Recommendations*: In this approach, only the absolute last actions performed by users are considered, regardless of their previous actions. This method is suitable in situations where the user's context dictates it, or where there is insufficient data to consider anything beyond the current context. For instance, if nothing is known about the user Ken, and he is watching a movie about cars on a Friday night, a similar movie might be recommended when he finishes. However, what isn't known is that Ken has been watching with his family, and after the car movie, his family goes to bed, and he wants to watch a documentary about the pyramids of Giza. Capturing this intent is challenging, but for the sake of argument,

[11] Some parameter names are adjusted for the sake of thesis continuity

[12] A powerset is the set of all possible subsets, if $I = \{1,2\}$ then the powerset of I would be: $\mathscr{P}(I) = \{\emptyset, \{1\}, \{2\}, \{1,2\}\}$

if it were known that around this time of night, users who usually watch car movies transition to documentaries, a recommendation for Ken's documentary could be made based on this last interaction and the time of day.

- *Session-Based Recommendation*: Rather than just considering the last interactions, all actions performed by a user in their current session could be considered[13]. For example, instead of just considering the last step of Ken watching a movie about cars, it is now known that, on that night, when Ken and his family sat down, he first searched for movies related to cars and then for documentaries, spending time reading the description of the Pyramids of Giza documentary. In this scenario, it is much more likely that a Session-Based recommender would pick up on these actions and recommend the documentary to Ken.
- *Session-Aware Recommendation*: Lastly, while considering the actions performed by the user in the current session, information regarding the user's past sessions might also be available. For example, it is known that every Friday, Ken watches a documentary after watching a movie about cars. Therefore, this weekend will be no different, and the latest documentary should be recommended to Ken. Here an important distinction has to be made, through this thesis i discuss Sequence Aware recommender systems, note that this is not the same as Session-Aware, Sequence Aware models are the superset or collective group of Last-N Recommendation systems, Session-Based Recommendation systems, and finally Session-Aware, being our user personalized Session-Based models.

It is not necessarily known if Ken wants to watch a documentary or not. However, the task of recommenders in context-adaptation scenarios is often characterized as "finding matching items" for a given session's beginning, without explicitly stating what a good recommendation would look like (Quadrana et al., 2018). It could however be instructed to solve one of three potential tasks, it could be directed to find alternatives for an item, compliments, or continuations. This latter point is of primary concern in this thesis, as I wish to create a continuation of the current context (series or movie being watched). In terms of the implementation of these different context adaptions, Last-N interactions are the most prominent in recent literature, followed by Session-based approaches and lastly Session-Aware implementations (Quadrana et al., 2018). The approach we use, presented in detail in Section 3.2.2, is a Last-N interaction-based approach, however before I can discuss this in detail, I must first discuss the algorithms employed.

2.2.2 Sequence Algorithms

Quadrana et al. (2018) identify three main classes of algorithms: *Sequence learning*, *Sequence Aware matrix factorization* and hybrids. For the scope of this thesis I am only concerned with *Sequence learning*[14] as covering all areas of of this research field is out of

[13] Sessions are often described in the context of web-browsing, being the time you spend on a certain site, a sessions could end when you close the site, but could also span a longer time if a cookie is set in your browser. I utilize viewing-sessions in this thesis which describes the time spent and actions done whilst watching a specific movie or show.

[14] Note that the majority of research is conducted using some sort of sequence learning

topic, but again refer to Quadrana et al. (2018) for further details. Sequence learning is in turn divided into sub-classes, of these *Frequent pattern Mining* and *Sequence Models* is most relevant, but other sub-classes are *Distributed item representations* and *Supervised models with sliding windows*.

Frequent Pattern Mining

Frequent Pattern Mining (FPM) techniques originate from the task of discovering user consumption patterns in large transaction databases. Patterns are traditionally put into three categories:

- Association Rule Mining (AR) (Agrawal et al., 1994), where we identify items that frequently occur together in the same transaction, regardless of the order.
- Sequential Pattern Mining (SP) (Agrawal & Srikant, 1995), where we identify a pattern only when the items occur in the same order.
- Contiguous Sequential Patterns (CSP) (Quadrana et al., 2018), where we further require that the co-occurring items are adjacent in the sequence of actions within the same transaction.

These patterns or rather, rules, are often pre-computed offline before any inference is made, e.g. considering some set of information, compute patterns of items as rules in regards to one or more of the three FPM techniques. This computation results rules with connected values often referred to as *confidence* and *support* (Quadrana et al., 2018), which express, for example, the strength of the rule. When conducting a prediction, a partial transaction is given, and the system should predict its continuation by looking up associated rules from the database and applying these to the partial transaction. For example, transactions are produced by users when they watch item j and then watch item i, resulting in the transaction $j \rightarrow i$, which is stored in a database. Given a set of thresholds, such as items needing to be watched on the same evening or being in the same genre, patterns are mined from these transactions. This process provides a resulting set of rules with values attached to indicate how confident the system is that a given rule is the best for the item. For instance, if Ken is watching a movie about cars j_1 (the partial transaction), and there is a rule $j_1 \rightarrow i_1$ with high *confidence*, the documentary i_1 can be recommended to Ken (see Figure 2.4)

Early research (Mobasher & Nakagawa, 2002; Mobasher et al., 2002), employed AR, SP and CSP in a web usage mining scenario, finding that AR and SP with their less constrained patterns, lead to better recommendation results, whilst CSR scored better when trying to predict which page the system should pre-fecth[15] for a user. In a more recent work (albeit still in 2012), Yap et al. (2012) showed promising results when weighing the patterns mined based on their relevance to the user the system is recommending for. Such an approach would make sense in the example above regarding Ken, given that the original rules mined considered all transactions, the chance of the majority transactions being the same as Ken's interests (e.g. cars → documentary would be rather low, so therefore, ranking the rules

[15] Imagine a google search result, when you enter the page of "cars", the recommender will try to pre-fetch the page that you're most likely to click on.

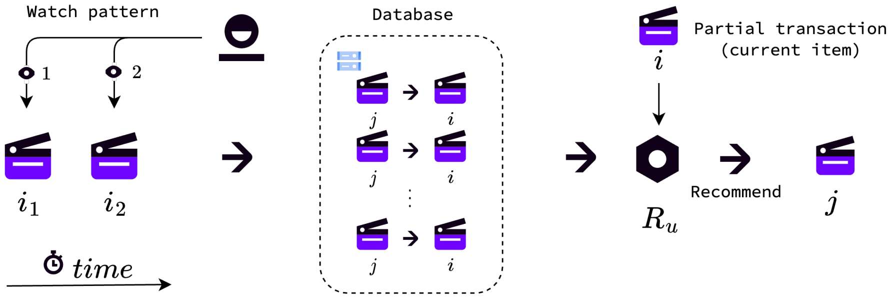


Figure 2.4: A simple example of a FPM technique.

based on e.g. those containing documentaries, which is known to be an interest of Ken, would lead to a more suitable result.

Frequent Pattern Mining methods are well explored and easy to implement and interpret (Quadrana et al., 2018). However, the problem of finding suitable threshold values (parameters) for the offline mining task is very domain-specific, and are alongside issues of scalability, some of the drawbacks of this approach (Quadrana et al., 2018). The main parameter is often the *minimum support value*, which states when something is supported as a rule, e.g. the time between movies before we say that it's a sequence. If this value is set too low, there might not be enough transactions to draw value from the data, and if set too high, it might result in a noisy rule-set, providing no real value.

Sequence Modeling

Sequence Modeling has been shown to produce good results in later years for Sequence Aware recommendation strategies (Quadrana et al., 2018). Models utilized for this are often categorized into three parts: *Markov models*, *Reinforced Learning* and *Recurrent Neural Networks*. There has been a fair amount of research in finding the best models to solve the issue of context adaptation, primarily in the E-commerce, Music and WWW domains (Latifi et al., 2021; Nasir & Ezeife, 2023; Quadrana et al., 2018; Wang et al., 2021). In later years the focus has turned more towards deep-learning approaches as they produce better results (Wang et al., 2021; Ludewig & Jannach, 2018), however some works have shown that more simple, Neighborhood models perform almost as well (Latifi et al., 2021). As an example Jannach & Ludewig (2017) found that a simple K-Nearest Neighbor (KNN)[16] model performed almost as well as a much more advanced neural model.

In 2018 Ludewig & Jannach (2018) benchmarked algorithms from literature in regards to Session-based algorithms, where they found that although there was a pattern of Neural models performing generally better, in different recommender scenarios, e.g. different datasets, the simple baselines AR, SR and Session-KNN performed in turn almost as well as the neural counterparts. Ludewig et al. highlighted this as a result of a lack of a

[16] Recall our example from the very beginning in regards to Ken, I mentioned looking at similar users, KNN is one approach where we consider the K, ergo the 3, 10, 20 nearest neighbours of Ken to find recommendations in.

standard evaluation suite for session-based algorithms but argued that this finding could be useful when considering which more complex algorithm to employ in a system, once the benchmark results for AR, SR and KNN approaches have been found. As an example, if the Session-KNN model which has a co-occurance[17] approach shows promising results, spending time and resources in employing a complex neural model which takes heavy precedence from the order of sequence items, might not be the best idea. This sentiment was somewhat echoed in regards to Sequence Aware recommender systems (Latifi et al., 2021), where another aspect was that baseline[18] algorithms often don't get enough attention and care as the new algorithm proposed, leading to a sort of confirmation bias[19]. In 2021, Wang et al. (2021) further echoed an issue regarding the cross-domain aspect of Sequence Aware recommender systems, stating that these algorithms are produced as a cross-domain solution, and by disregarding domain-specific aspects, the results might not be directly transferable between domains.

I will not delve deep into all the different models found in Sequence modeling in this thesis, but recommend Latifi et al. (2021); Ludewig & Jannach (2018); Wang et al. (2021) for a good overview and explanation. I do need to cover two essential models however, these being the Simple Associated Rules (AR) model Agrawal et al. (1993) with a maximum rule size of two, A simple First-Order Markov Model (MC) as implemented by Ludewig & Jannach (2018), and finally a Sequential Rules (SR) model introduced by Kamehkhosh et al. (2017).

Simple Associated Rules (AR) is a model based on the Associated Rules FPM concept, where co-occurrences of items are mined in a set of sessions S, given the last entry point $|s|$ and a target item i, the model is formalized by Jannach & Ludewig (2017) as:

$$\begin{aligned}\mathrm{score}_{AR}(i,s) = {} & \frac{1}{\sum\limits_{p\in S_p}\sum\limits_{x=1}^{|p|}\mathbf{1}_{\mathrm{EQ}}(s_{|s|},p_x)\cdot(|p|-1)} \\ & \sum_{p\in S_p}\sum_{x=1}^{|p|}\sum_{y=1}^{|p|}\mathbf{1}_{\mathrm{EQ}}(s_{|s|},p_x)\cdot\mathbf{1}_{\mathrm{EQ}}(i,p_y)\end{aligned} \tag{2.6}$$

Where S_p is the set of all sessions p, s is the current user session, i is the item I want to score for and $\mathbf{1}_{EQ}(x,y)$ is a function returning 1 if x and y are equal, and 0 otherwise. Although daunting at first, the model is rather straight forward. The left hand side of the formula, more specifically the part beneath the denominator: $\sum_{p\in S_p}\sum_{x=1}^{|p|-1}\mathbf{1}_{\mathrm{EQ}}(s_{|s|},p_x)\cdot(|p|-1)$ simply counts the number of times the last action of a sequence $s_{|s|}$[20] appears in all other sequences in the collection S_p. This ensures that the values are normalized in terms of the global count of the event for item $s_{[s]}$. The right side of the model simply checks if the item

[17] If two items appear in the same session, (As in Sequential Pattern Mining), we consider that a viewing session for a user, without caring for the order of these items.

[18] Baseline algorithms are used as a comparison for the algorithm you're employing.

[19] A confirmation bias is a bias where one subsequently only consider the results that matches the goal of your hypothesis, e.g. if I have two models, and spend a ton of time fine-tuning one, and disregarding the other, the results would be biased as more or less rigged the experiment to fit my story.

[20] $|s|$ refers to the *cardinality* of the set s, also known as the amount of items in s, e.g. if $s = \{1,2\}$ then $|s| = 2$, so when we index s at position $s_{|s|}$, we simply state that we want the final item in the set of s.

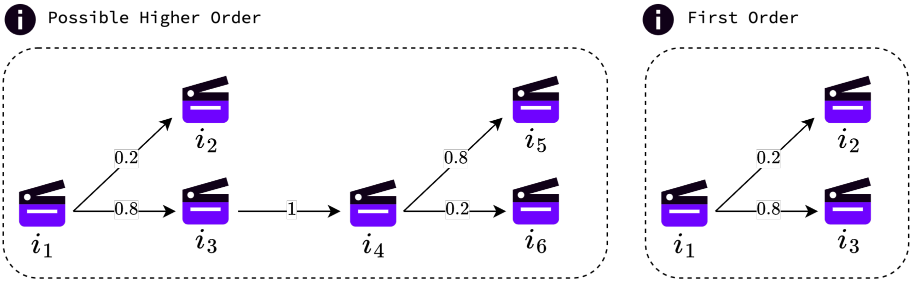


Figure 2.5: An examaple of a Markov Chain.

i occurs in the same session as the last entry in the user session $s_{|s|}$, in other words, it checks for any co-occurance.

Although this is only a formalization and not a statement of the actual implementation of an *AR* algorithm, the normalization aspect of the model, ergo the lefthand side, would also punish any items appearing in longer sessions, due to how we multiply by $(|p|-1)$ in the denominator. This was most likely done to assure that sessions with only one item/event would not count towards the normalization.

Markov Chains (MC) is an approach introduced in Ludewig & Jannach (2018) which can be seen as a variant of AR with focus on the sequences in the data, rather than the co-occurances. It is based on a first-order Markov Chain, where the items follow a contiguous sequential pattern. A Markov chain can briefly be defined as a mathematical system that undergoes transitions from one state to another on a state space, where the probability of each next state depends only on the current state and not on the sequence of states that preceded it (Norris, 1998). In simpler terms, Markov Chains are sequences of items, where the probability of a transition between two items is only dependent on the current item (Markov state), and the subsequent items in the branching chain. Consider Figure 2.5, a First-Order Markov Chain would only ever care about the first step in the chain, e.g. from item i_1, only consider the transition probabilities i_2 and i_3. A Higher order Markov Chain, considers the transition probabilities between items across a larger chain, such as the transition probability between i_1 and i_5, multiplying the subsequent probabilities. The model is formalized as:

$$\text{score}_{MC}(i,s) = \frac{1}{\sum_{p\in S_p}\sum_{x=1}^{|p|-1} \mathbf{1}_{\text{EQ}}(s_{|s|},p_x)} \sum_{p\in S_p}\sum_{x=1}^{|p|-1} \mathbf{1}_{\text{EQ}}(s_{|s|},p_x)\cdot \mathbf{1}_{\text{EQ}}(i,p_{x+1}) \tag{2.7}$$

Here, the left-hand side of the formula is also a normalization step, where the global occurrence of the last entry in the user session $s_{|s|}$ is counted if $s_{|s|}$ is not the last entry in the session p[21]. The right-hand side is a simple count of instances where the pattern $s_{|s|} \rightarrow i$

[21] Basically, counting the occurrence of the rules $s_{|s|} \rightarrow any_i$

is found.

2.3 Hybrid Recommender systems

Combining different recommender systems have shown to continuously increase the accuracy of predictions, and are a part of the latest trends in recommendation strategies (Jannach et al., 2010; Roy & Dutta, 2022; Aggarwal et al., 2016; Ricci et al., 2011). There are several Hybrid strategies for different use cases, one often applies one type of hybrid strategy over the other based on the goals of the recommender, be that mixing Collaborative data with knowledge data, or as in the case of this thesis, mixing collaborative data with sequential data. Burke (2002) originally provided a taxonomy detailing 7 different hybrid strategies, but one can generally abstract these into three categories, *Monolithic*, *Parallelized* and *Pipeline hybrids*. In regards to this thesis, the concern is with the Parallelized approach, more specifically a weighted hybrid.

As defined in Jannach et al. (2010) a weighted hybrid strategy combines the recommendation of two or more recommender systems by computing weighted sums of their scores. It can be formalized by the following:

$$R_{weighted}(u,i) = \sum_{k=1}^{n} \beta_k \times rec_k(u,i) \tag{2.8}$$

where item scores need to be restricted to the same range for all recommenders and $\sum_{k=1}^{n} \beta_k = 1$. These two latter statements are important in terms of a weighted hybrid approach, as they ensure that item scores from different algorithms are compared correctly. Seeing as the scores can originate from two different systems, one could be on a range of 1-10, whilst another 0-1, normalization techniques are often applied such as a min-max normalization method, where we force both scores into the same range, or a softmax approach which given a vector of item scores, will provide a new score for each item corresponding to the likelihood of that item having the highest score. If you sum up all the scores for these items, it will equate to 1. If a softmax normalization strategy is applied, it is critical that both vectors of item scores has the same amount of items, if not it will lead to a bias towards the shorter vector, again due to how the probabilities will sum to 1. See the comparative Tables 2.1 and 2.2 for an example of how scores would change through a softmax normalization strategy.

Table 2.1: Softmax results for 3 items.

Item Scores	**Softmax Probability**
2.0	$\frac{e^{2.0}}{e^{2.0}+e^{1.0}+e^{0.1}} \approx 0.65$
1.0	$\frac{e^{1.0}}{e^{2.0}+e^{1.0}+e^{0.1}} \approx 0.24$
0.1	$\frac{e^{0.1}}{e^{2.0}+e^{1.0}+e^{0.1}} \approx 0.11$

Table 2.2: Softmax results for 5 items.

Item Scores	Softmax Probability
0.5	$\frac{e^{0.5}}{e^{0.5}+e^{-0.2}+e^{0.0}+e^{1.2}+e^{-1.5}} \approx 0.28$
-0.2	$\frac{e^{-0.2}}{e^{0.5}+e^{-0.2}+e^{0.0}+e^{1.2}+e^{-1.5}} \approx 0.16$
0.0	$\frac{e^{0.0}}{e^{0.5}+e^{-0.2}+e^{0.0}+e^{1.2}+e^{-1.5}} \approx 0.20$
1.2	$\frac{e^{1.2}}{e^{0.5}+e^{-0.2}+e^{0.0}+e^{1.2}+e^{-1.5}} \approx 0.33$
-1.5	$\frac{e^{-1.5}}{e^{0.5}+e^{-0.2}+e^{0.0}+e^{1.2}+e^{-1.5}} \approx 0.03$

Further, $\sum_{k=1}^{n} \beta_k = 1$ indicats that the sum of the weights has be 1, in the formula provided, β_k is utilizes to indicate the weight for recommendation model R_k, this is often also indicated with w_k, and is what I denote in this thesis. To provide a more explanatory formalization of the strategy, consider recommendation model 1 and recommendation model 2, denoted as R_1 and R_2 respectively. If the goal is to promote recommendations made in R_2, the weight of R_2, w_2, should be set to 0.7. Since all weights must sum to 1, the weight for R_1, w_1, is therefore 0.3. Obtaining a recommendation for user u on a single item i from both algorithms can therefore be stated as:

$$R_{weighted}(u,i) = R_1(u,i) * w_1 + R_2(u,i) * w_2 \tag{2.9}$$

Often, however, the goal is not only to score a single item but to score a set of items from each recommender. For example, when querying two recommenders for their top k items, these scores can be combined to obtain the top k, or rather the *argmax* of the two resulting sets:

$$rec_{weighted}(u,i) = argmax^{k}_{i \in R1_{u,i}, R2_{u,i}}(R1_{u,i} * w_1 + R2_{u,i} * w_2) \tag{2.10}$$

Parallelized designs such as the one above require careful tuning of weights to produce good results (Jannach et al., 2010), another aspect is how these values are normalized, something I will discuss further in the methods section of this thesis.

2.4 Movie Domain

The domain one is trying to solve for, often dictate the methods, obstacles, and objectives of the recommender. The Movie and TV domain is no exception, and are met with several issues of their own. The domain of this thesis could arguably be the TV Domain, however I declare the Movie domain. The TV Domain is often broader than just the content, considering not only the TV programs such as movies or shows but also TV Channels, Commercials, Educational TV shows, etc. (Véras et al., 2015). Notes are taken from both domains, however this is not declared as a cross-domain recommender approach, as it does not necessarily exploit any of the user-centered aspects found in, for example, the TV domain, which is noted as one of the characteristics of cross-domain recommender systems (Fernández-Tobías et al., 2012). These characteristics would have been more relevant if the goals were aligned within the context of recommending TV Channels[22] as was done by Bambini et al. (2011) in relation to IP TV recommendations, then utilizing the characteristics of the Movie domain to recommend in the TV domain would have been more applicable. The models, evaluation strategies, information-mining and previous work are all based in the Movie domain, the only exception to this is the algorithm architecture, as in the modern day and age with the rise of IP TV technology, the lines between traditional Broadcasted TV and video on demand services are ever blurring.

In 2006 Netflix provided the academic community with *The Netflix Price* (Bennett et al., 2007), a challenge accompanying a dataset, to which the prize was one million US dollars. The task was simple, improve the Netflix algorithm by 10%. After three years, eventually Töscher & Jahrer (2009) hit the score threshold, and the competition came to a close. The dataset Netflix provided for this challenge, known as the Netflix Price dataset, consisted of *explicit feedback* from users in the shape of ratings, on a scale 1-5. It was at the time the largest dataset provided to the academic community, and through this dataset, over 20 thousand teams signed up for the competition. The Netflix price has been accredited lighting the spark for research in the movie domain of recommender systems. Together with the MovieLens datasets (Harper & Konstan, 2015) the community had ample datasets to conduct their research on, and perhaps due to this availability, the Movie Domain has (Véras et al., 2015) and is still (Roy & Dutta, 2022) one of the more active domains in recommender system research.

It is therefore not surprising that there are a lot of publications in the movie domain. In the aftermath of the Netflix prize competition in 2009, latent factor models were seen as state of the art (Véras et al., 2015; Aggarwal et al., 2016; Koren et al., 2022). In later years, the focus shifted to one of three main focuses, optimization, hybrid-recommender approaches, and deep-learning approaches (Roy & Dutta, 2022). Discussing all of the latest work in the Movie domain is out of scope for this thesis, however for a more in depth literature review I recommend the work done by Véras et al. (2015), which in-depth covers the TV Domain up till 2015, catching a lot of the work done in the movie domain up as well, and Roy & Dutta (2022), detailing the latest developments in the recommender system literature [23].

2.5 Architectural implementation

When conducting experiments online in a production grade system, understanding how the recommendation came to be is key to understanding results. I put emphasis on the implementation of our recommender in this thesis, and therefore need to discuss the architectural aspects of recommender systems. To the best of my knowledge Véras et al. (2015) was the first to include the architectural implementation in a literature review of recommender

[22] Remember that TV 2 is a commercial public broadcaster first and foremost

[23] As the Movie domain is arguably the largest domain, a regular recommender system survey is bound to catch most of the interesting developments

systems in the Movie/TV domain, and I wish to add to this by reporting our implementation in Section 3.4.2. Some of the more recent issues facing recommender systems as entailed by Roy & Dutta (2022), are all linked to the system implementation. Gathering good implicit feedback is all about measuring client-side, and storing, aggregating, and processing said feedback to make it available for the model to learn. Handling real-time user-feedback is also strongly connected to how one processes the signals from the client, but is primarily an algorithmic concern. Finally, how well one can measure system performance is bound to the architecture more than anything.

Based on the categorization provided by Coulouris et al. (2005), six categories of architectural implementations can be distinguished.

- *Stand-alone*: The recommendation is computed and served on a single device, such as e.g. a model embedded in the memory of your iPhone, although discredited in literature for RS approaches, such a method might make more sense in the generative AI space, e.g. embedding a small language model on your mobile device, a concept which was not thought feasible at the time of the literature reviews writing.

- *Client-Server*: Is where the client sends a request to a server with the users' preferences, which has been processed on the client. The server in turn handles the request, conducts the recommendation, and returns a set of items to the client, it is based on the *fat client* concept where computational load is partly shared between Client and Server.

- *Web*: The Web architecture shares most of the same characteristics as Client-Server, but is based on the *thin client* concept. The client is only responsible for serving the data to the user and communicating the user's feedback to the server, the server then, in turn, performs the computational tasks, such as conducting a recommendation and building the user preferences.

- *Cluster*: A cluster architecture also follows the same preface of the client-server approaches, but the computational load of producing the recommendation is distributed across several computation nodes tightly connected, and an example of such an application can be found in Aljunid & Manjaiah (2019), where the Collaborative filtering model is computed in a spark cluster, allowing for impressive parallelisation of the computational strain.

- *Cloud*: A cloud based architecture is not the same as a cluster based architecture, but they are closely related. Whilst the computational strain may be distributed across a cluster as well, this is but one aspect of the larger system, where some parts of the systems can be concerned with computing the user profile, another might be tasked with storing the user history, viewing-sessions, etc. Another key distinction is in the way the nodes are connected. In a cluster architecture, the nodes of the cluster are tightly connected, most often in a closed sub-network, whilst a cloud architecture can be widely distributed, so that e.g. a compute node in the US is responsible for recommending a user from Iowa items, whilst another node in the UK is responsible for the user from London.

- *P2P*: The Peer to Peer Architecture model is not commonly used in recommend system applications (Véras et al., 2015), but some works found the model to increase the recommendation service availability (Han et al., 2004; Wang et al., 2008). This model differs from the client-server approach due to how each node in the distributed system acts as both a client and a server.

In 2015, the most popular architectural implementation was the *Client-Server* implementation, (Véras et al., 2015). A lot has changed since then with the availability and knowledge of cloud services increasing, and with the current trends of cloud compute dominance, any serious recommender system is most likely deployed through a cloud architecture. Although both Web and Client-server might rely on cloud architecture in terms of a single node processing the recommendations, one node will never be able to handle the vast amount of requests that a modern service receives. One such example is the Pinterest recommender system (Liu et al., 2017), where they at first utilized a cluster architecture based on Hadoop[24], but having to move over to a different cloud based architecture due to the sheer magnitude of requests, showing that even sharing the load across a single cluster is not enough.

2.6 Related works

Due to the wide availability of datasets such as the MovieLens dataset, a fair few sequential recommendation models have been introduced as a good choice for the movie domain, providing us with arguably a lot of related work. This cross-domain aspect was stated by Wang et al. (2021) as a possible limitation however, due to how focusing and optimizing for one domain might produce better results rather than introducing a catch all solution. Further, the modern neural models, such as GRU4REC (Hidasi et al., 2015), HGRU4REC (Quadrana et al., 2017), Caser (Tang & Wang, 2018), AttRec (Zhang et al., 2018) were all evaluated through offline evaluation, and not necessarily tested and fine-tuned for the Movie domain. Although the Caser model performed better than GRU4REC, and the AttRec model, in turn, beat out Caser, the GRU4REC model was shown to have minimal improvements over a traditional KNN model in Bernardis et al. (2022). They implemented a set of neural models, amongst these the GRU4REC and a KNN model in a Video on Demand recommender system through an offline evaluation scheme, where they found minimal improved results for the neural models, but as in Jannach & Ludewig (2017), finding that the KNN approach performed almost as well, at a fraction of the compute cost. The models introduced in this thesis are akin to the baselines used in these various papers, such as Markov Chains (Ludewig & Jannach, 2018). In 2019 Zeng et al. (2019) introduced their novel approach of Bidirectional Item Similarity (BIS), and further the Adaptive Bidirectional Item Similarity (ABIS) for the next recommendation task in the movie domain, showing yet again an improvement over the neural models in the offline evaluation scheme. In terms of similar work conducted in live environments, the only point of reference we could find was the paper by Zhao et al. (2019) where they showed

[24] https://hadoop.apache.org/, Hadoop is primarily used as a distributed storage system, but was used to find and count co-occurring image pairs through a MapReduce job, check out https://apache.github.io/hadoop/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html for a deeper understanding.

improvements for the next-item recommendation task in regards to the Youtube[25] platform by improving upon a deep neural network-based point-wise ranking model, introduced in Covington et al. (2016), to take into considerations more than one objective for the next-recommendation task.

Although most of the related work shows that neural models are superior to simpler alternatives, the objective of this thesis has to be taken into account. I am deploying an algorithm to the TV 2 Play platform, and do not have the scores of data scientists available to implement and finetune a deep learning algorithm, therefore it is critical to iteratively find what works, before committing to a neural approach. There are complications with deploying neural algorithms, as stated by Sculley et al. (2015), further in our case of Youtube, they first implemented more rudimentary matrix factorization approaches (Covington et al., 2016), which they, in turn, benchmarked up against the neural models, stating that whilst gains are first found through offline evaluation, it is the A/B results of an online evaluation that make or break the iteration. Another case is that of Pinterest (Liu et al., 2017), where they showcase the evolution of recommender systems on their platform, starting from a simple co-occuring MapReduce job, and transforming into a similar ranking-based algorithm as is employed by Youtube.

The final aspect which makes my thesis different from the aforementioned Sequence Aware approaches, comes from the nature of movies, the goal of the recommendation strategy, and the dataset available to me. Although some previous approaches utilized movie domain centric datasets such as the MovieLens dataset to evaluate the different models, how a session is defined based on this dataset varies. The highly used Movielens dataset is constructed of a set of timestamped ratings, indicating when a user has rated a movie, and not necessarily when a user has actually watched the movie, this is the first abstraction. Further, in some cases, the considered temporal differences between when a user interacted with an item are not considered in minutes or seconds, as in Zeng et al. (2019) where the distance is based on the order of items, rather than how much time is in between items. This is the second abstraction. Finally, the models are often generalized to work for several datasets, however, the difference in the considered content, that being a song, a movie, or an item on an e-commerce shop, is also not directly considered, leading to our third and final abstraction. In papers such as Ludewig & Jannach (2018) and Quadrana et al. (2018) the MovieLens dataset is not considered at all, although these papers broadly cover available datasets and methods in the Sequence Aware recommender system field. To the best of my knowledge, this thesis is the first to employ a Sequence Aware recommender approach in the movie domain, without any of these three abstractions of the data.

[25] https://www.youtube.com/

Chapter 3

Methodology

In this section I detail the methodology employed to reach our research goals. This includes an overview of the datasets employed, the methodology for both offline and online evaluation, an overview of the platform design employed to perform a recommendation in the live environment, TV 2 Play, and finally a section connecting the methods proposed to our research questions and objectives.

3.1 Datasets

As the models used in this thesis are based on two different recommender system strategies, such as matrix factorization and frequent pattern mining, two different datasets are required. The models are trained on different aggregations or arrangements of the same core viewing data, something I elaborate on in Section 3.4, however downloading and processing this core viewing data to conduct analysis and perform offline evaluation would not be feasible due to the sheer magnitude of data. To be able to answer the research questions, I therefore download the processed dataset `viewing_data` containing the necessary information to conduct matrix factorization, and the dataset `viewing_sequence_data` which is extracted from the core viewing data so that I might evaluate our models. Due to the business-sensitive nature of the data, I am not able to report on direct viewing counts or impressions, but will approximate or utilize percentages where applicable.

Before delving into the details of the dataset, a few aspects have to be addressed. TV 2 Play is a streaming service providing access to a large variety of content, this includes movies, series, live streams, clips and news. One part of the TV 2 terminology is the concept of `asset` and `categories`, where an asset can be a movie or an episode of a series, and a category can be either a season of a show, the show itself, or even a category of movies. In this thesis, I am interested in the show itself or the movie, and not the individual episode of a show, this has implications which I will discuss later. To make this distinction apparent, `item` is formalized as either the movie itself, of the show title, ergo if a user has watched an episode of Farmen, a dataset oriented around the item will have the ID of the show (category) Farmen, and not the episode (asset). Another distinction is between Users and User profiles, like any other modern streaming service, a single user can have several profiles attached to that account. Since different user profiles can be argued to belong to different people, e.g. members of a family with different interests, I perform recommendations on the user profile level, and not user level. When discussing users in this thesis I therefore refer to the user profile.

3.1.1 Viewing data

The dataset `viewing_data` shows what and for how long a user has watched an asset on TV 2 Play, aggregated on item. This means that if a user watched three episodes of the latest season of Farmen, which are all 45 minutes long, the result is a row with that users id, the id of the show Farmen, and a duration of 135 minutes. Further, the datset has the field `contentType` which indicates if this is a movie, series, livestream, clip, etc., `kids` indicate if this content is kids friendly or not, `appName` indicates the app the user viewed the content on, this can be Chromecast, TV OS, Android TV, Desktop, etc., `durationSec` is how long the user watched this item measured in seconds, `assetIds` is the list of assetId's that user watched, and finally `firstStart` and `lastStart` indicates when the user first started watching, and when they last pressed start. See Table 3.1. The dataset consists of data from 6 months, starting in November, and ending in April, there are more than 2 million active user profiles in this period across more than 60 million rows, when aggregated on item level, the dataset contains more than 5000 unique items (series and movies). Based on the dataset, it is possible to distinguish between what content type the users are watching, how much they are watching, and for how long. Users watch an average of 1.5 items per day. Note that this is somewhat biased due to the aggregation of series watching. Therefore, the `assetId` column is exploded, resulting in an average assets watched metric of 6. The average `durationSec` is ≈ 9000. As is shown in Figure 3.1, series dominate, following by movies and then sport (`MATCH`).

Table 3.1: Viewing Data Sample Dataset.

Field Name	Example Value
userId	...
profileId	...
itemId	604034768
contentType	SERIES
kids	False
appName	...
durationSec	32369
assetIds	[a, b]
firstStart	2024-04-01 12:47:09
lastStart	2024-04-02 15:51:03

Further, the data is plagued by some popularity bias, with approximately 60% of all watching conducted on the top 25 items (see Figure 3.2). This bias is also evident in terms of `durationSec`, where 54% of all watching is conducted on the top 25 items (see Figure 3.4). This latter number drops to 50% when live channels (`LIVECHANNEL`), which are inherently biased, are filtered out.

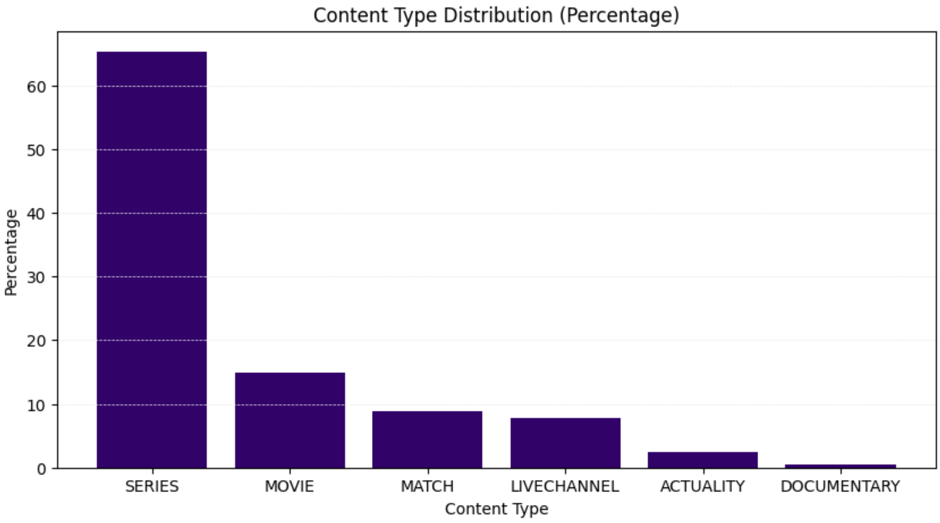


Figure 3.1: Content Type Distribution.

For the sake of this thesis, I decided to restrain the recommendations to the `SERIES` and `MOVIES` content types, this reduces the dataset by 20%. The data is aggregated on the user and item level to establish a one-to-one relationship between users and items, reducing the dataset by an additional 16%. Further filtering is applied to entries where `durationSec` is less than 500 seconds, and a score ceiling is set at 5000 seconds, removing another 8%. Note that this filtering is done after aggregating on the user ID, so if a user watched 400 seconds one day and 400 the next, the combined value of 800 seconds would pass the filter. The rationale behind setting this upper score ceiling was to avoid series dominating the user-item matrix. In the end I have reduced the dataset to close to 30 million rows. Finally, BM25 weighting is applied to the data, but I cover this more in detail in Section 3.2.1.

3.1.2 Viewing Sequence Data

The Markov model as described in Section 3.2.2 is dependent upon mined rules based on a set of criteria. There is a lot of data in TV 2, and if I were to perform rule mining for e.g. 6 months of data each time I wanted to train the model, it would result in an unnecessary amount of compute power. In Section 3.4 I describe in detail how this compute load is mitigated by storing the aggregated rules between items on a daily basis, however, to conduct analysis and offline evaluation I need a more granular selection where the individual sequences of items are presented. I extract one month of data where stating that I want all sequences of data if and only if two items are watched after one another by the same user, within a time frame of 20 minutes. This results in approximately 35 million sequences.

Firstly, I introduce the term *Binge*, as a sequence where the two items in our sequence are part of the same show. On the TV 2 Play platform, the algorithms are only ever shown when there are no more episodes of the same series to be shown, if a user is watching episode 1 of Farmen, the system will not show the user the next-poster where my

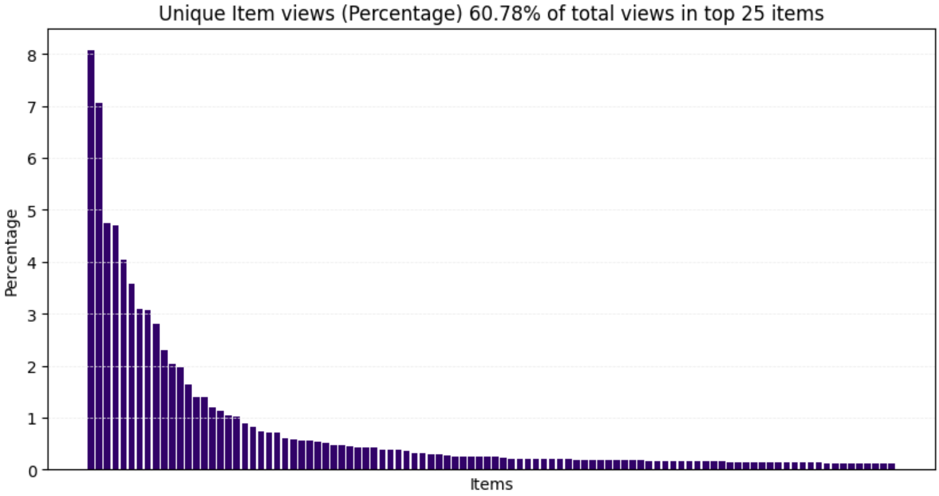
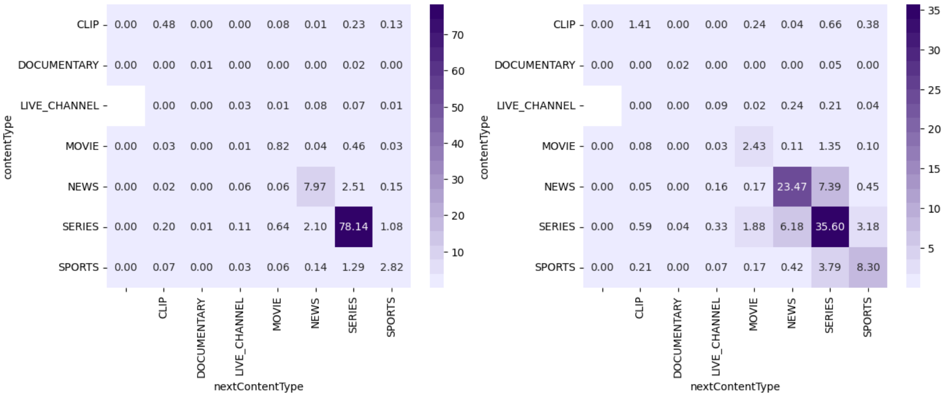


Figure 3.2: Unique item views (Percentage) on top 100 items

algorithm is presented, but rather a "play next episode" interface where the next episode in the series automatically plays. Not only are self-recommendations not of interest in the model, but to adapt to the context of the user, automatic play indications should also be disregarded from the recommendation, as it is not an action done by the user, but rather a lack thereof. The source of this data is the raw watch data from TV 2. Unlike the CF dataset, which is already conveniently arranged by item, this dataset requires enrichment with the necessary `item_id` information. This results in the columns `item_id`, `next_item_id`, `profile_id`, `measure_date`, `content_type` and `next_content_type`. Removing all binge items reduces the dataset significantly by 70%, resulting in approximately 10 million item sequences. When considering the content type of the item sequences, it is observed that before removing binge, almost 80% of all sequences are of the type `SERIES`→`SERIES`, with `NEWS` taking the second place at 8% (see Figure 3.3a), after removing binge however, the sequences are more distributed, with `SERIES`→`SERIES` still taking first place at 35% (see Figure 3.3b). Due to the restriction of the CF model to only recommend Series and Movies, the sequences are also restricted to these two content types. This results in a further 50% reduction, leaving approximately 6 million sequences.

3.2 Algorithms

Following the objective of designing a hybrid model that combines the broad user preferences of Collaborative Filtering and the contextual relevance of a Sequence Aware model, I choose to implement a Collaborative Filtering model based on Matrix Factorization to find user preferences and a simple Markov model to try and adapt the context of the users on a more general scale. Both are rather rudimentary models, but combined they can serve to solve the next-poster problem. I chose to utilize the collaborative filtering model due to the simple fact that this model was already established in the TV 2 System, I do report



(a) With Binge (b) Without Binge

Figure 3.3: Heatmap of transissions between content types, recall that Binge when a transition is between the same item.

the details of this model but do not conduct any hyper-parameter optimization, rather trusting in the configuration found in TV 2. For contextual relevance, I chose a Markov Model approach due to several factors. Firstly and most importantly, due to the objective of deploying and evaluating this model in a real industry environment, simplicity matters. Further, based on literature the also arguably simple KNN models was considered as an alternative, due to how they proved strong competitors to neural models in Jannach & Ludewig (2017) and Bernardis et al. (2022), however Ludewig & Jannach (2018) reports that the Markov model outperforms the KNN approaches in the evaluation schemes most alike the problem case in this thesis, that being the @3 list length evaluation cases on media datasets[1]. Further in Zhang et al. (2018) the Markov model introduced proved a competitive baseline in terms of the MovieLens dataset, although there are no KNN alternatives to compare here. Finally, since movies and series are the central content for this thesis, how the sessions are interpeted matters greatly. Whereas previous research worked with datasets containing several events for each session for each user, there are very few cases where a user watches three movies in a row. If I were to only consider a Sequence Aware approach, abstracting the sessions as has been done in previous research could have been applicable, e.g. where I look at all movies a user watched over a period of time of days and weeks, and build a linear viewing pattern based on that, however, I believe we can adapt the context better by using contiguous sequential patterns. The argument is based on the fact that just because a user watched one movie one day, and then another the next day, this does not equate to the movies being watched after one another. Abstractions are well, but the ground truth is preferable when available. With these aspects in mind, I chose the Markov model rather than the KNN approaches. Finally, the Hybrid approach was chosen based on the idea that if an item is recommended to the user based on their preferences, and also based on the adapted context, that item should be recommended and if there is no overlap, the

[1] These results are found in the appendix of the thesis, note that this is not a direct translation to our problem case, but is a strong indicator.

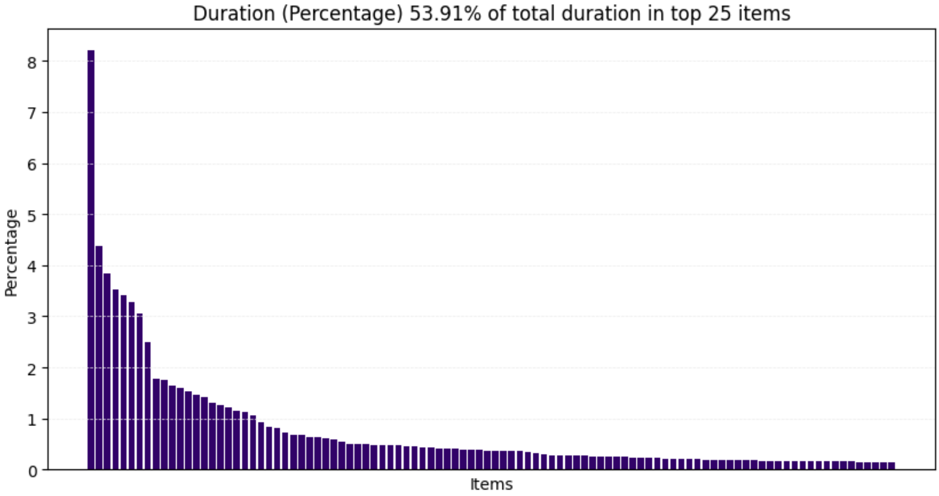


Figure 3.4: Duration, (how much time a user spent on an item) (Percentage).

item scoring best based on both models should be recommended.

3.2.1 CF Model

The Collaborative Filtering model applied in this thesis is a traditional ALS implementation from the Implicit python library. The implementation is almost identical to the one introduced in Hu et al. (2008) As detailed in Section 2.1.3, the goal is to find the factor $q_i \in \mathbf{R}^f$ for every item i, and the factor $p_u \in \mathbf{R}^f$ for every user u in our dataset. These factors are found by minimizing function (3.1) for every possible observation[2] (u,i):

$$\min_{q_*,p_*} \sum_{u,i} c_{u,i}(r_{u,i} - q_i^T p_u)^2 + \lambda \left(\sum_u \|p_u\|^2 + \sum_i \|q_i\|^2 \right) \tag{3.1}$$

Recall that $\lambda \left(\sum_u \|p_u\|^2 + \sum_i \|q_i\|^2 \right)$ is the regularizing aspect of the model, where λ is used to adjust the magnitude of observations, here with the addition of adding together the L2 Norm for every occurrence of the user and item in question. Further, when working on implicit data the aspect of confidence comes into play. Simply because a user has not seen an item, or interacted with it, does not mean that the user dislikes that item. Hu et al. (2008) introduces the term confidence which can be briefly explained as "how much do we know that the user likes this item", it has a general formalization as:

$$c = 1 + \alpha r_{u,i} \tag{3.2}$$

Where α is set as a parameter to the model, e.g. if its set to 40, the default confidence score for all items would be $1+40$. In my implementation, α is fixed to 1, whilst the regularization

[2] Recall that when applied to implicit data, we no longer only care about the known ratings $(u,i) \in \mathbf{K}$, but all possible ratings

variable λ is set to 0.1. Recall that I discussed the ALS approach as alternating between first fixing p_u, solving for q_i, and then in turn fixing q_i solving for p_u. The model is trained through 30 such iterations, finding for 32 latent factors. See Table 3.2 for an overview.

The CF implementation in this thesis differs from Hu et al. (2008) through a pre-processing step known as BM25 weighing. BM25 weighing, originally introduced in Robertson & Sparck Jones (1994); Robertson et al. (1995), originates from the field of information retrieval and search engines to estimate the relevance of documents to a given search query. The version of BM25 that has been applied in the Implicit library is more akin to the simplified version proposed in Robertson et al. (2004).

Although there is no direct statement in the implicit documentation regarding the exact formalization, the implementation can be seen in Code Snippet A.2. The denoted `K1` (k_1) and `B` (b) parameters from the function call are the free parameters in the BM25 method. k_1 controls the non-linear term frequency effect, while b controls the document length normalisation (Robertson et al., 2004). In regards to this case, as watch time is used as implicit feedback, a high k_1 will give more importance to higher watch times on individual items, while b, on the other hand, adjusts the impact of the number of items a user has interacted with. A high b would penalize users who have interacted with many items. An argument for a high b is that it prevents "super users" from having too much of an impact in the recommendation. Together with the code snippet from above and the formalization from Robertson et al. (2004) BM25 can be formalized as:

$$\mathrm{BM25}(i,u) = \frac{f(i,u)\cdot(k_1+1)}{f(i,u)+k_1\cdot\left(1-b+b\cdot\frac{|u|}{avg(|u|)}\right)}\sum_{i\in \mathrm{I}}\mathrm{IDF}(i) \tag{3.3}$$

Where i is the item, u is the user, $f(i,u)$ is the frequency of item i for user u, $avg(|u|)$ is the average number of items per user, $|u|$ is the number of items interacted with by user u, I is the set of all items, and IDF is further denoted as:

$$\mathrm{IDF}(u) = \log\left(\frac{N-n(i)+0.5}{n(i)+0.5}+1\right) \tag{3.4}$$

Table 3.2: Parameters used in the ALS model.

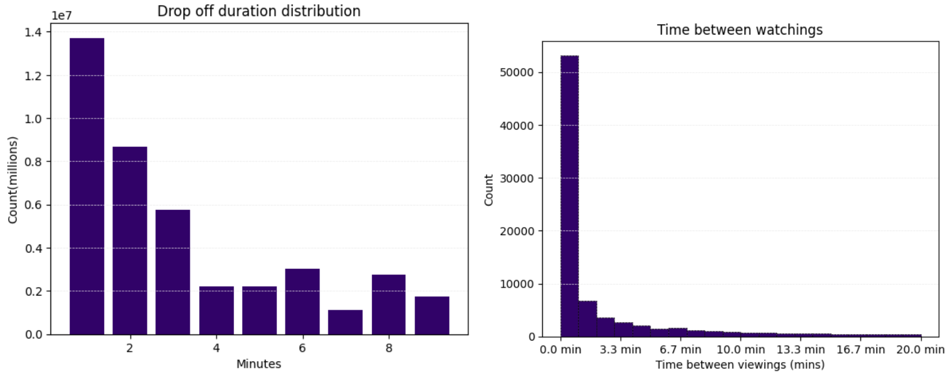
Parameter	Value
f	32
λ	0.01
$iterations$	30
α	1.0
cg	False
k_1	3.0
b	1.0

The parameters are reported in this section (as seen in Table 3.2) because this model was inferred from the TV 2 Play recommendation system as the baseline for the experiment.

No further adjustments were made through either offline or online evaluation schemes, and no new hyper-parameter search was conducted for the model, trusting in the already established standard. The model is trained each day on data from 2022 to the current date.

Through computing the minimizing function (3.1) we are left with our feature vectors $q_i \in \mathbf{R}^f$ for every item i, and the factor $p_u \in \mathbf{R}^f$ for every user. Through these we can score for any user u and any item i a predicted score $\hat{r}_{u,i}$ through the dot product of the feature vectors:

$$\text{score}_{CF}(u,i) = q_i^T p_u \tag{3.5}$$



(a) Drop off duration distribution.

(b) Time between viewings

Figure 3.5: Drop off duration and time between viewings, both relevant for setting the restrictions on rule mining.

3.2.2 Markov Chains

The second model is the Markov model introduced in Ludewig & Jannach (2018). More formally however, the model is a Last-1 Interaction based recommender, mining contiguous sequential patterns, through a first-order Markov approach. The model is originally formalized to take into account the case of an item i and a user's session s, however, in this thesis, the model is not inferred with a given session, as only the Last-1 interaction is considered as an item j. Therefore, the model is formulated as:

$$\text{score}_{MC}(j,i) = \frac{\sum_{p \in S_p} \sum_{x=1}^{|p|-1} \mathbf{1}_{\text{EQ}}(j, p_x) \cdot \mathbf{1}_{\text{EQ}}(i, p_{x+1})}{\sum_{p \in S_p} \sum_{x=1}^{|p|-1} \mathbf{1}_{\text{EQ}}(j, p_x)} \tag{3.6}$$

Recall that S_p is the set of all sessions p, $\mathbf{1}_{EQ}(x,y)$ is a function that returns 1 if $x = y$ and 0 otherwise. All scores for any rule $j \rightarrow i$ can be achieved by processing once across the dataset, which in turn can be stored in a hashmap[3] for easy inference. The model will

[3] A hashmap is a data structure where given a key k you can retrieve an associated value v, a well known implementation of a hashmap is the Python Dictionary data structure.

Table 3.3: Parameters used in the Markov model.

Parameter	**Value**
s_{max}	12 min
w_{min}	3 min
i	a
j	b
$days$	180
$normalization$	log_2
S_p	{...}
I	{...}

return the transition probability between two items, based on a set of conditions. Recall that deciding upon these conditions are very important for the model, and in this case it is no different. Two parameters are defined: s_{max} and w_{min}. s_{max} is the maximum amount of time between two viewings in the same viewing session s. For example, if a user first watches item i_1 and then starts watching item i_2 5 minutes later, s_{max} must be set to greater than 5 minutes for the session to count towards the rules. w_{min} is set to 3 minutes because it is observed that after 3 minutes, the likelihood of a user dropping out of the asset stabilizes[4] (see Figure 3.5a). For the s_{max} parameter, a value of 12 minutes is decided upon. There are arguments for increasing and decreasing this latter parameter, but ultimately after passing 12 minutes the overall length between sessions stagnates (see Figure 3.5b). Finally, the number of days to consider for the training set of the model is important, leading to the introduction of the parameter $days$. The full parameter list can be seen in 3.3.

Another important aspect of the model lies in the data of which it is built. Whilst the sequence data is not as heavily affected by popularity as the base viewing data, this popularity is still present, and has some implications. When simply computing the transition probabilities based on the count of rules, situations can arise as shown in Figure 3.4 where two very popular shows have an extremely high transition probability, leading to a very long tail of low scores. If I were only interested in scoring for the top item, or the top sequence of items, this would not be an issue. However, due to the intention to combine this model with the CF model, a more distributed score is needed. Otherwise, no matter how the hybrid model is adjusted, the top item here will always score best. Various options were explored, but a strategy of performing log2 normalization on the counts was chosen, before calculating the transition probability. This approach leads to a much more even top-20 result. With this in mind, the model is reformulated as:

$$\text{score}_{MC}(j,i) = \frac{\log_2\left(1+\sum_{p\in S_p}\sum_{x=1}^{|p|-1}\mathbf{1}_{\text{EQ}}(j,p_x)\cdot\mathbf{1}_{\text{EQ}}(i,p_{x+1})\right)}{\log_2\left(1+\sum_{p\in S_p}\sum_{x=1}^{|p|-1}\mathbf{1}_{\text{EQ}}(j,p_x)\right)} \tag{3.7}$$

[4] This was found by querying the viewing times of users on an asset from the TV 2 Viewing database.

Table 3.4: Example of Log_2 Normalizartion.

item_id	next_item_id	count	frequency	log2_count	log2_frequency
604001423	604010361	21064	0.003339	14.362492	0.000032
604001423	604005881	6116	0.000969	12.578373	0.000028
604001423	604023261	4523	0.000717	12.143064	0.000027
604001423	604005262	3664	0.000581	11.839204	0.000026
604001423	604045897	2445	0.000388	11.255619	0.000025
604001423	604001861	2372	0.000376	11.211888	0.000025
604001423	604024104	2161	0.000343	11.077483	0.000025
604001423	604022261	2117	0.000336	11.047806	0.000025
604001423	603999723	1188	0.000188	10.214319	0.000023
604001423	604004261	1019	0.000162	9.992938	0.000022
604001423	604046736	1008	0.000160	9.977280	0.000022
604001423	604032101	988	0.000157	9.948367	0.000022
604001423	603999607	578	0.000092	9.174926	0.000020
604001423	604046168	570	0.000090	9.154818	0.000020
604001423	604037674	528	0.000084	9.044394	0.000020
604001423	604033562	501	0.000079	8.968667	0.000020
604001423	604034050	475	0.000075	8.891784	0.000020
604001423	603999486	444	0.000070	8.794416	0.000020
604001423	604010761	282	0.000045	8.139551	0.000018
604001423	603999421	249	0.000039	7.960002	0.000018

3.2.3 The Hybrid Model

To combine the scores of the two models I utilized a weighted hybrid approach, where I combined the CF model with the Markov model, to produce the top k items to the user based on item j that they are currently watching. The main goal of the hybrid approach is to recommend items that are 1. Relevant to the user, and 2. Adapting the Context of the user by recommending an item which they are likely to watch after the current item. The *CF* model returns a list of k elements, sorted after how relevant it is to the user based on the user's watch pattern, whilst the *MC* model returns the transition probability from any item j to all other relevant items i. These are inherently not necessarily comparable scores, which is why I need to perform normalization's operations such as $softmax$ before applying our weights. Given a set of top n scores from the CF model as $R^n_{u,i}$, and the set of top n scores from the Markov model as $B^n_{j,i}$ I formalize the hybrid model as:

$$H_{u,j} = argmax^k_{i \in I}(softmax(R^n_{u,i}) * w_1 + softmax(B^n_{j,i}) * w_2) \tag{3.8}$$

The model $H_{u,j}$ finds the top k items from the set of items I, based on the user u and the item j. The n parameter states how many items to consider from each recommender. Recall that the two sets of recommendations need to be the same length due to our normalization strategy.

One adherent issue with the Hybrid Model lies in the normalization aspect, due to the softmax normalization technique, I have to have the same length of items, e.g. it is not possible to combine a list of 5 and a list of 10 items fairly, as the list of 5 will dominate the recommendation. What happens then when the recommendation system is faced with a user of which there are no previous watch history for? Or if the item j, the item the user is currently watching, is brand new and there are rules for this item? This issue is referred to as the Cold-start issue in literature (Jannach et al., 2010). In an offline evaluation, the quick solution is to filter users and items affected by this cold-start issue, but in a live environment, simply ignoring the user or item is not feasible, leading the the need for a baseline alternative for this scenario.

3.2.4 Baseline Popularity Model

The mitigation to the cold-start issue in TV 2 Play is to simply present the user with a popularity model, as I will discuss further in Section 3.4. This popularity model is based on the principle of popular right now, where every 300 seconds the system calculates which items have been watched by the most unique users, over the span of 900 seconds. Such an approach has been introduced in literature before and shows promising results (Ludmann, 2017). The model cannot be directly compared to the other models due to how it is exposed to a lot less impressions, but it can imply the performance of popularity on the platform.

Table 3.5: Parameters used in the ALS, Markov and Hybrid models.

Model	**Parameter**	**Value**
ALS	f	32
	λ	0.01
	$iterations$	30
	α	1.0
	cg	False
	k_1	3.0
	b	1.0
MC	s_{max}	12 min
	w_{min}	3 min
	i	a
	j	b
	$days$	180
	$normalization$	log_2
	S_p	{...}
	I	{...}
H	n	20
	k	3
	w_1	0.5
	w_2	0.5

3.3 Research Design

To recap some of the objectives, I wish to implement and test my model in a real-world environment, utilize offline evaluation techniques and insights to prepare for this real-world environment, and analyse the performance of this model. Finally I also wish to compare the online and offline results, and therefore need some common metrics across the two testing environments. I start by describing the offline evaluation suite in Section 3.3.1, then online evaluation suite in Section 3.3.2.

3.3.1 Offline evaluation

The goal of the offline evaluation is firstly to find the best parameters for the Hybrid model, and secondly to perform an offline evaluation of said model alongside the two models it is built upon so that I might compare the results to previous work, see which model might affect the scores, and to compare the result with the online evaluation. I will start by discussing the metrics utilized in this thesis, and then detail how I implemented the models in the offline suite.

Metrics

Due to the hybrid nature of the model, I could have taken inspiration from both collaborative filtering approaches and Sequence Aware approaches for the offline evaluation, however as the next-poster problem is arguably a sequence problem, I only consider those relevant for the problem statement. The nature of the data also takes precedence, in Sequence Aware approaches the general task is to generate a list of objects based on a given item, how we find if this continuation is good or not, is again highly dependent on the application and the problem at hand. In my case the problem is to provide the user with an item that they are likely to *want to watch next*, this echoes the problems found in Sequence Aware approaches, where the typical approach is to withhold certain items of a session from the training set, and measure how well the algorithm is able to guess these "hidden" items (Ludewig & Jannach, 2018). As many of the models found in literature are trained to guess the *next immediate item* given the first n elements of the session, a popular evaluation method introduced by Hidasi et al. (2016) is to iteratively increment the length of the scored list of items k, measure Hit Rate (HR) and Mean Reciprocal Rate (MRR), and finally calculate the average MRR and HR across the k different lengths. MRR and HR are metrics originating from the Information Retrieval field Aggarwal et al. (2016), where HR is defined as:

$$\mathrm{HR} = \frac{\text{Number of Hits}}{\text{Total Number of Recommendations}} \tag{3.9}$$

where the number of hits is calculated, in this case, how many times the recommender recommended the hidden item at any position for a user, and divide that by how many recommendations were made in total. Jannach et al. (2010) declares hitrate more nuanced for an individual user as

$$\mathrm{hitrate}_u = \begin{cases} 1 & \text{if hits}_u > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

which is then in turn divided by the recommendations made for all users. This metric is rather naive in terms of rank however, meaning that when considering a recommender list of 3 items, if the "hidden" item provided by the test case was recommended at position 3, it would score equally to one where the "hidden" item was provided as the first item in the recommendation. MRR is presented as a mitigation to this, where the score is based on the rank it appears in, Aggarwal et al. (2016)[5] defined it as:

$$\text{MRR} = \frac{1}{|R|} \sum_{n=1}^{|R|} \frac{1}{\text{rank}_n} \tag{3.11}$$

Where $|R|$ is the count of all recommendations made, and rank_n is the position the "hidden" item was recommended at.

I am also interested in some additional quality factors, as one reflection by Ludewig & Jannach (2018) was that although the Markov models performed well in some cases where with shorter list lengths, ergo k, this was often accompanied by high popularity, and low coverage. Further, as was seen with the `viewing_data` dataset, there was a high indication of popularity bias in the training data. With this in mind and following previous research, I therefore also report on *Coverage* and *Popularity*. *Coverage* is dependent on how many unique items appear in the top-k recommendations for all users, also referred to as *aggregate diversity* by Adomavicius & Kwon (2011). It can simply be formulated as:

$$\text{Coverage} = \frac{|\text{Unique Items in Top-K Recommendations}|}{|\text{Total Unique Items in Catalog}|} \tag{3.12}$$

For *Popularity*, as in Ludewig & Jannach (2018) I report the *Average Popularity Score*(APS) as:

$$\text{APS} = \frac{1}{k} \sum_{n=1}^{k} \text{popularity}(i_n) \tag{3.13}$$

Where k again is the length of the recommended list, and popularity is the *individual popularity score* for the item i_n. In previous research this popularity score is based on how often any item i appears in any session from the training set, this can be formalized as:

$$\text{count}(i) = \sum_{s \in S_s} \sum_{x=1}^{|s|} \mathbf{1}_{\text{EQ}}(i, s_x) \tag{3.14}$$

Where S_s is the set of all sessions in the training set, and $\mathbf{1}_{\text{EQ}}(a,b)$ is a simple function that returns 1 if $a = b$ and 0 if not. This count is then normalized through min-max normalization to get the scores on a scale of 0-1, which in turn can be formalized as:

$$\text{popularity}(i) = \frac{\text{count}(i) - \min(\text{counts})}{\max(\text{counts}) - \min(\text{counts})} \tag{3.15}$$

Note that so far I have discussed popularity in regards to the session training dataset, however as I are operating with two datasets, I can introduce different popularity scores for any item, one originating from the sessions as stated above, and one based on the overall

[5] Note that they call it average reciprocal hit rate (ARHT).

viewing on the platform, which I define as *General Average Popularity Score* (GAPS) for the sake of this thesis. GAPS is equal to APS, up until the count function, redefined as general count:

$$\text{count}(i) = \sum_{j \in R} \mathbf{1}_{\text{EQ}}(i, j) \tag{3.16}$$

Where R is the set of user-item ratings from the `viewing_data` dataset and j denotes the rated item j. Recall that the CF model is trained on the *durationSec*, ergo how much a user has seen an item, an interesting reflection would be to see if the model is more prone to recommending items which has been watched a lot, rather than just frequently watched, so I therefore finally introduce the popularity metric *General Duration Popularity Score* (GDPS), which is defined as:

$$\text{GDPS} = \frac{1}{k} \sum_{n=1}^{k} \text{d\_popularity}(i_n) \tag{3.17}$$

where *d_popuarity* in turn is defined as:

$$\text{d\_popularity}(i) = \frac{\text{duratoin}(i) - \min(\text{duratoins})}{\max(\text{duratoins}) - \min(\text{duratoins})} \tag{3.18}$$

where durations is the set of all `durationSec` and finally duration is defined as:

$$\text{duration}(i) = \sum_{u,j \in R} j_{\texttt{durationSec}} \tag{3.19}$$

Offline Evaluation Suite

I perform a Grid Search to find the most optimal configuration for the Hybrid model H and Markov model MC, but do not explore the different parameters for the collaborative filtering model CF, as these parameters are set from the industry standard in TV 2. I evaluate the models on the dataset `viewing_sequence_data`, which is data from the month of April to the start of May, and therefore train the models on data from November, up until March. For the CF model this equates to the `viewing_data` dataset that I explored earlier, however for the MC model I have to borrow some data from the TV 2 system to train locally. I briefly mentioned in Section 3.1.2 how I store the aggregated rules between items on a daily basis in the TV 2 system, and will describe this process in detail in Figure 3.7, but for now: due to how I store the aggregated rules, I can simply extract this data from TV 2, reducing the amount of data drastically, as I only store how many rules, e.g. transitions between and item i_1 and an item i_2 that happened that day. Given the same time frame, these aggregated rules would be the same as if I took the resulting 6 million sequences in the `viewing_sequence_data` dataset and counted the unique rules. The parameters considered in the grid search can be seen in Table 3.6, if a parameter is omitted, it is the same as the one defined in Table 3.5.

To score the metrics I sample 1000000 cases from the `viewing_sequence_data` dataset, and hold out the first item in the sequence, i_1, evaluating the resulting list of recommendations provided by the model. The evaluation suite is coded in Python, and is available in the thesis linked github[6] repository.

[6] https://github.com/sfimediafutures/MA_Snorre-Alvsvaag

Table 3.6: Parameters considered for grid search.

Model	Parameter	Value
ALS	f	32
	λ	0.01
	$iterations$	30
	α	1.0
	cg	False
	k_1	3.0
	b	1.0
MC	s_{max}	12 min
	w_{min}	3 min
	$days$	$[180, 30]$
	$normalization$	$[log_2, none]$
H	n	$[20, 50, 100]$
	k	$[1, 3, 5, 10, 20]$
	w_1	$[0.1, 0.3, 0.5, 0.7, 0.9]$
	w_2	$[0.9, 0.7, 0.5, 0.3, 0.1]$

3.3.2 Online evaluation

The goals of the online evaluation is to measure the performance of the Hybrid model in a real-life environment, in a set of configurations, so that I can analyze the results and compare them to the offline evaluation approaches. To achieve this, I firstly need an online evaluation suite, allowing me to benchmark the model, and some metrics that the two evaluation strategies would have in common. Thanks to the collaboration with TV 2, I am able to run a field test for the recommender strategies on the TV 2 Play Platform[7], and for metrics, I am able to measure Click Through Rate (CTR) and MRR for accuracy, and a set of session duration milestones to gauge the user engagement with the content they clicked on. Whilst these latter session duration metrics are isolated to the online evaluation, they are critical for any real indication of performance. As argued by Jannach & Zanker (2022) CTR, whilst a strong indication of short and long user preference, optimizing primarily for short term CTR can have negative business impact in the long run, due to aspects such as click-bait alienating the customer. Further Zheng et al. (2010) noted that there is a trade-off between optimizing item rankings for clicks, and optimizing it for the relevance to the customer. With this in mind, I define and state our metrics.

Metrics

The vast amount of available data in TV 2 would allow me to perform different accuracy and user satisfaction metrics, however, the scope is limited by the technology stack which reports viewing data back to the system. I discuss the details of the data pipeline in Section

[7] https://play.tv2.no/

3.4, but to be able to measure the different metrics I employ a set of different SQL queries towards the TV 2 system, which aggregate the data to daily granularity, further grouped by the rank of the showed items, (e.g. we show 3 items, therefore we have one day, with one row for rank 1, rank 2, and rank 3 with impressions and clicks). With this in mind I am able to calculate the accuracy measures CTR and MRR, and further an indication of user satisfaction through various viewing milestones which are reported back by the client. I state that it is an indication of user satisfaction, as a direct measurement of user satisfaction is primarily concerned with explicit feedback through more qualitative measures such as an online survey (Jannach et al., 2010). MRR is already defined in the offline evaluation section, but CTR is defined as:

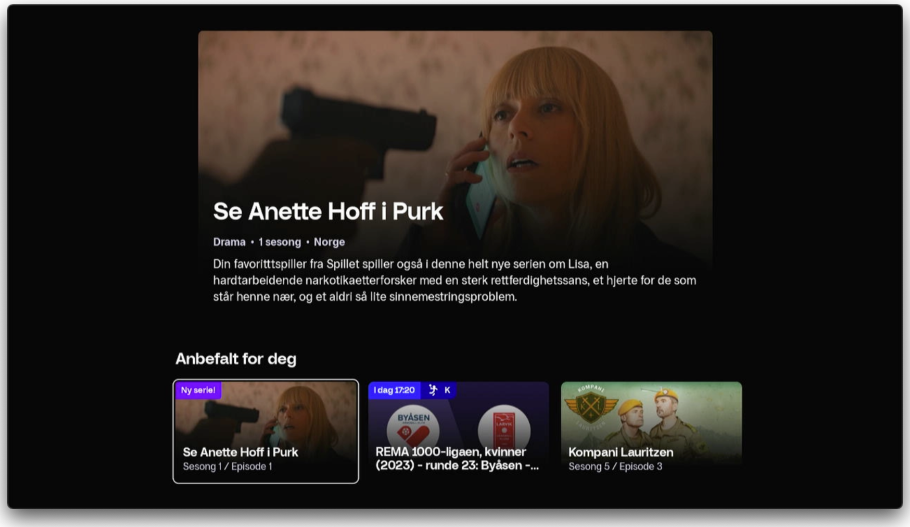
$$\text{CTR} = \frac{\text{Number of Clicks}}{\text{Number of Impressions}} \tag{3.20}$$

CTR has its root in information retrieval and indicates based on a list of items presented to the user (impressions), which if any items the user clicked on (number of clicks). In terms of recommender system applications, it is often reported in the E-commerce domain (Jannach & Zanker, 2022), but also in live applications of video on-demand platforms such as Youtube (Zhao et al., 2019). In this application, CTR is equal to Hit Rate, recall that hitrate was defined as number of hits divided by the total number of recommendations made for any user, and in regards to the nature of this thesis, given a session, there will only ever be one click per session.

The aforementioned viewing milestones are reported as a count of how often a click led to the user watching a certain amount of the provided item, showing the click's success, and indicating the users' satisfaction with the item they chose. I report four different metrics for this, $V@1min$, $V@2min$, $V@3min$, $V@50\%$. I also further report on the popularity and coverage metrics introduced in the offline evaluation section, importantly here however are the considered items the popularity score is calculated on. For continuity with the offline evaluation, where the popularity scores were based on the training data, I consider the last 5 months of viewing data with the latest date being the day before conducting the online evaluation.

Online evaluation suite

The online evaluation suite is a live environment across all Android TV's watching TV 2 Play. Whenever a user has finished watching a movie, or a show where there are no more episodes of that show, they are presented with the next-poster, see Figure 3.6, where the user is presented with three possible items that they might find interesting, to keep the user engaged, or rather to recommend them *something to watch next*. I conduct an A/B test, where the recommended items shown in these three positions are either A, the Hybrid model, or B, the baseline *CF* model. Several baselines were considered for this experiment, such as an Item-Item recommendation strategy, but ultimately due to the limitation of only being able to conduct an A/B test, and not an A/B/C, I chose the weathered *CF* model already established in the system. Upon reaching the next-poster, the client sends a request for a recommendation, and based on the ID of the user they are provided with either A, the hybrid model, or B the *CF* model. I discuss the technical implications of this in the next



Se Anette Hoff i Purk

Drama • 1 sesong • Norge

Din favoritttspiller fra Spillet spiller også i denne helt nye serien om Lisa, en hardtarbeidende narkotikaetterforsker med en sterk rettferdighetssans, et hjerte for de som står henne nær, og et aldri så lite sinnemestringsproblem.

Anbefalt for deg

Figure 3.6: Screenshot of the next-poster in action, above we see a preview of the first item presented after the end-credits finished. The second item is football match, and the third is the popular show Kompani Lauritzen. This example is editor-controlled, hence why there is a football match recommended.

Section.

Finally, due to the live nature of the system, there are cases where the system is faced with a new user who has not generated enough data to be considered by the *CF* model, or cases where a new movie is published, where there aren't enough data to provide any next recommendations for this item. This case is the aforementioned cold-start issue. In TV 2 the standard is to default to a basic most popular recommendation in these cases, which is what we also do in our experiment presenting the user with the baseline Popularity model introduced in Section 3.2.4. The recommendation will be defaulted to the popular model P if and only if either the Markov model MC or the Collaborative Filtering Model CF has less than n possible items. Recall that n is the parameter in the hybrid model deciding how many items are considered from both recommenders and that due to the softmax normalization strategy utilized, getting the same amount of items from both recommenders is important for a balanced recommendation. Whilst the result of the popular model might not be directly comparable with the Hybrid and *CF* model due to how it is treated as a fall-back, and not given the same amount of impressions, it can show implications of popularity, and I, therefore, report it alongside the two other models.

With these objectives in mind I wish to conduct at least one experiment, evaluating the best-performing model based on the offline evaluation. However, due to how previous research has shown that models scoring well in offline scenarios don't necessarily score well in online scenarios (Beel & Langer, 2015; Garcin et al., 2014; Maksai et al., 2015; Rossetti et al., 2016), I also wish to add two more configurations of the model to be able to assess if the offline evaluation is able to indicate the performance of the online evaluations. One of these I argue is the "basic" implementation of the model, weighing the CF and MC model equally, the other differs from the best-performing model in terms of how many days the MC model has been trained on, and the n parameter, increasing this to 100 to account for the increase in possible rules. I included this latter to firstly aid in the coverage and popularity aspect of these models, but also due to how the $days = 30$ configuration led to overall higher scores, wishing to see if I could replicate this online. The different configurations are:

- **Configuration 1**, denoted H_1. This is the best configuration for the Hybrid model applicable in the online evaluation, with the parameters: $n = 20$, $w_1 = 0.1$, $w_2 = 0.9$, days = 30 and $normalization = log_2$. This configuration scored best for both MRR and HR.
- **Configuration 2**, denoted H_2. This is the more default configuration, where: $n = 100$, $w_1 = 0.5$, $w_2 = 0.5$, days = 180 and $normalization = log_2$.
- **Configuration 3**, denoted H_3. This has the best scoring weight configuration but with more training data and a higher n value, where: $n = 100$, $w_1 = 0.1$, $w_2 = 0.9$, days = 180 and $normalization = log_2$.

I will report fully on limitations in Chapter 6, however one important limitation is the restrictions of conducting A/B testing in TV 2 Play, where I can only consider two cases side by side at one time. This is a limitation of the TV 2 System at the time of writing, and due to this, I tested each case over a set period, against the baseline CF model. Choosing to spend the available time in unison with my objectives, I prioritize H_1 and H_2, lending H_3 only three days of evaluation. Reporting on the exact number of users exposed to the different algorithms is not feasible due to the business-sensitive nature of the platform, however, note that each model is exposed to more than a hundred thousand unique users.

Finally, TV 2 Play is a complex system, and conducting a recommendation is no easy feat, there are many aspects of this system that can influence the resulting recommendation beyond the model itself, such as viewing history, an influx of new data and users, and last but not least system performance. I will delve into this system in the following section and will explain in part how a recommendation goes from user feedback to items in a TV.

3.4 Platform Design

TV 2 Play is a video streaming platform serving over 2 million user profiles domestically, built around a micro-service architecture. Micro-service architectures allow for flexibility and scalability, but also brings forth a lot of complexity. The streaming platform itself is a large system, and the recommender system utilized in support of this system is no different. In this Section I will cover parts of this system, in Section 3.4.1 I will discuss aspects relevant to data gathering, whilst in Section 3.4.2 I will showcase how the recommendation models introduced in this thesis are deployed and maintained in a cloud-based architecture.

3.4.1 Data gathering

When working in a large environment such as TV 2, gathering and processing the data is in and of itself, key for any meaningful analysis and recommendation strategies. TV 2 Play has apps and players on most TV's, handheld devices, and app stores in the western user market[8], they even have an app for the Tesla car. Such a large set of apps and players requires a robust system underneath to capture, process, and store viewing data, impressions, etc. Discussing this system in detail is outside the scope of this thesis, however I will discuss some aspects relevant to data gathering, aggregating, and cleaning of the data used in the implementation of the Hybrid algorithm in the TV 2 Play apps.

Viewing data in TV 2 is processed through a combination of Apache Spark[9] and Apache Flink[10]. Apache Spark is a well established tool for large-scale data processing, allowing for distributed data processing at scale, whilst Apache Flink is a framework and distributed processing engine for stateful computations over unbounded and bounded[11] data streams. Another important component of the pipeline is Apache Kafka[12], Kafka is an event streaming platform, and we often discuss kafka-streams, or kafka-topics. The two are practically the same, but imagine that a kafka-stream is an unbounded stream of data, just like water in a pipe, whilst the topic is the tube of which the data is flowing through. When I discuss a kafka-topics, I refer to this tube. The system rests upon Google Cloud Storage[13] (GCS), where processed data is stored in Buckets. A storage bucket is a container for storing objects, which can be any type of data such as files, images, videos, backups, or documents.

Data originates in the video players embedded in the different client-apps. The apps utilize both Google Analytics[14] and Snowplow [15] to fire events which are stored en mass in a storage platform known as Google BigQuery[16], which are then composed to sessions by a Flink job based on events sent by those players, or in some cases a scheduled SQL query. These sessions are in turn aggregated into different configurations by a set of Spark jobs, varying in what information is stored alongside the sessions, and the time granularity. One important configuration variance is the `by_item` and `by_asset` distinction. As TV 2 Play operates with a large proportion of TV Series, they are some times interested in the season, episode, or series as a whole, e.g. if we wish to recommend on a series basis rather than an episode basis, such a distinction is key. The Collaborative Filtering model is trained on data from one such spark job, whilst the Markov model is trained on another which I introduced to the system, examples of this can be seen in Figure 3.7.

[8] TV 2 Play serves Norwegian customers primarily, however the app-store ecosystem in Norway is oriented around the typical western markedplaces.

[9] https://spark.apache.org/

[10] https://flink.apache.org/

[11] Unbounded and Bounded data streams refer to if the stream has a known end, e.g. an unbounded data stream has no known end to it, and would be interpreted by a system to continuously deliver data indefinitely.

[12] https://kafka.apache.org/

[13] https://cloud.google.com/storage?hl=en

[14] https://marketingplatform.google.com/about/analytics/

[15] https://snowplow.io/

[16] https://cloud.google.com/bigquery/docs/introduction

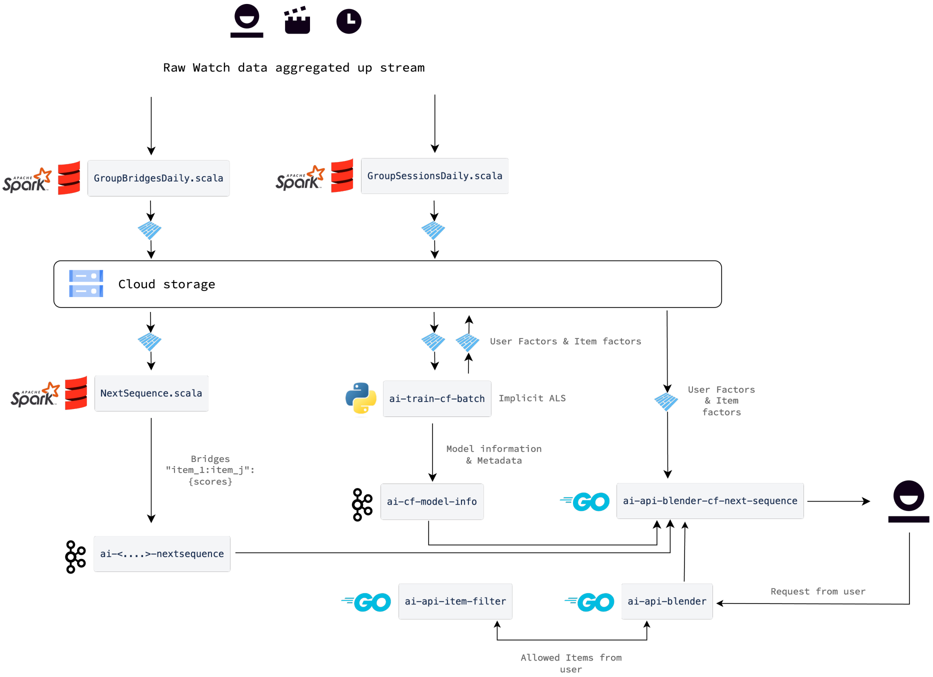


Figure 3.7: The system architecture, showing the training and inference pipelines of the TV 2 recommender system.

3.4.2 Recommender System Overview

The recommender system set in production in TV 2 Play is built on a cloud based recommender architecture, with several different applications responsible for different aspects of the recommendation process. This micro-service architecture allows us to utilize the best tools for the job, processing data with Spark, training our models with Python, and serving them through a set of Golang[17](Go) applications. Both Spark and Python are no strangers to recommendation system pipelines, but Go is not as explored, although there are examples of Go being utilized for the recommendation task (Fan et al., 2019). In this section I will explain in detail how the TV 2 play recommendation system is deployed, and what this thesis contributed to this system. As entailed above, data is aggregated and stored in a storage bucket through a set of Spark and Flink jobs, this data is in turn accessed by different spark jobs to perform the needed aggregations, filtration's, and transformations needed to train our models. Note that Spark is not necessarily a programming language. Rather, it is a powerful data processing engine that can be used through various programming languages such as Python (using PySpark), Scala, Java, and R. In TV 2 these jobs are primarily written in Scala[18]. The relevant system architecture is visualized in 3.7, which in turn can broken down into three sections: The Collaborative Filtering training pipeline, the Markov training pipeline, and the Inference pipeline. Firstly I will cover the relevant information regarding our target medium for this thesis, Android TV.

The Clients

Where you consume a recommendation matters (Adomavicius & Tuzhilin, 2011), be that on your phone, laptop or TV. As mentioned TV 2 Play is available on a lot of platforms, and each specific platform needs a certain type of application. Examples of these platforms are iOS Mobile, iOS iPad, Android Phone, Android Tablet, Apple TV, Android TV, the list goes on. All of these needs their own dedicated app to be developed, although they can share codebases in some cases. The important point here is that implementing a feature across a wide array of applications might not be beneficial if you don't know to which extent that feature actually increases the value of your product. This development cost is why I am only testing the next-poster recommendation strategy on Android TV, as this covers a vast amount of users due to its integration's in popular TV brands such as Sony, Samsung and Philips. Note that this limitation lies on the Client side, and not the recommender system side, although there is no guarantee that the best model configuration scores the same across different mediums e.g. TV vs mobile.

The Collaborative Filtering Training pipeline

Recall that the matrix factorization approach is a two step process, we first compute the minimizing function (3.1) leaving us with the feature vectors $q_i \in \mathbf{R}^f$ for every item i, and the factor $p_u \in \mathbf{R}^f$ for every user u. We then compute the dot product between these to two and get the predicted score $\hat{r}_{u,i}$. This two step process allows us to perform

[17] https://go.dev/

[18] https://docs.scala-lang.org/tour/tour-of-scala.html

the different aspects of the models at different times, once we have the feature vectors, we don't need to compute the dot product at once. The pipeline starts with the spark job `GroupSessionsDaily.scala` running and aggregating the latest viewing data as described above, the files are then stored to GCS in a parquet[19] file format, after this the python job `ai-train-cf-batch` is run with an accompanying algorithm configuration, declaring any filters applicable to the data such as the `contentType` field we discussed earlier. When the model is complete it first saves the joint latent matrix $\mathbf{R}^f$ found through our matrix factorization back to GCS, then produces a message on the kafka-topic `ai-cf-model-info` with information regarding the model, such as which filtration's were applied, when it was run, the path to where the features are stored on GCS, and most importantly the mapping tables needed to map user ID's and item ID's to the indexes of our joint feature matrix. This method allows us to load this now trained model into any application which in turn would be interested in using this specific model, e.g. one application could be responsible for the pure CF recommendations, and another might be responsible for a Hybrid approach, however both rely on the same core model. By using Kafka like this the system is also event driven, where any application can listen for new models on this topic, and load it as it is available, ensuring that the latest user interests and latest items are available as fast as possible.

The Markov training pipeline

The Markov training pipeline is much alike the CF pipeline, with the exception that the entire model is based on Scala Spark code. As mentioned before the Markov model requires more granular data than the CF model, but by taking advantage of the different parts of the model we are able to optimize the entire process. Firstly by mining all rules[20] for a specific day, we can store these rules along with their frequency, reducing the total amount of sessions processed each learning cycle to those within a one day window. The Scala Spark `GroupBridgesDaily.scala` is responsible for this aspect, and is run on a daily schedule, storing the aggregated rules to GCS in a parquet file format. It is this job where we utilize the s_{max} and s_{min} parameters to decide when to store a rule. After this we schedule the next job `NextSequence.scala`, which calculates the probabilities based on the n last days of data. The example code snippet for this can be found in the Appendix under Code Snippet A.1

Which provides us with the rule $i_j \rightarrow i_n$ and the accompanying scores needed such as the `frequencyScore` and `frequencyScoreNormalizedLog2` for every item j to any other relevant item i. These "bridges" as we dubbed them in the code are sent as a message to the kafka-topic `ai-prod-nextsequence`, so that any application in need of the scores between any subsequent items can simply load the kafka-topic into memory, as we will describe below. Here one important aspect needs to be addressed, for each subsequent run of `NextSequence.scala`, we produce a new set of scores for any item j and any item i based on the *days* parameter accounting for the newest transitions mined, which will alter the transition probabilities between all items. Kafka deals with keys and values just like a

[19] Parquet is an open source file format built to handle flat columnar storage data formats, much like CSV, but accredited for its performant compression, see https://parquet.apache.org/.

[20] recall that a rule is a transition from one item $i_1 \rightarrow i_2$)

traditional hash map, and in the same way Kafka expects us to provide it with unique keys. The individual Kafka topic can be configured to overwrite identical keys and only keep the latest provided message, allowing us to simply send a new message with the unique key, and the subsequent new scores. The data itself looks like this:

```
"itemid_j:itemid_i": {
    frequencyScore: 0.1,
    frequencyScoreLog2: 0.2,
    ...
}
```

where the key is `"itemid_j:itemid_i"`, allowing us to keep track of duplicate entries, and only keep the latest scores from any item j to any other item i. One limitation of an application such as this is also related to the nature of Kafka, if we want to reduce the amount of days considered for the `NextSequence.scala` job, we could end up still keeping some of the rules from the broader time window, if the transition $j \rightarrow i$ was not present in the data. Whilst not a critical aspect, its worth to note.

The inference pipeline

With both the models trained and ready to recommend, we need to consider two final factors. The hybrid recommendation, and the inference from the clients. This final pipeline is the larger of the three, and encompasses a set of services written in Go, each with a distinct job, and the clients themselves. In a live environment there are several aspects and business requirements to consider, some items might not be available any more due to licenses running out, perhaps the user profile is a kids profile, and other times the user might have a certain subscription resulting in them not having access to certain premium content. Whilst these are not necessarily the recommendation system's responsibility, we need to take these into account and assure that the recommender system either follows these requirements, or is deployed in such a fashion that it wont circumvent them. Finally we do not want to recommend a user an item they have already watched, whilst such a reminder strategy has been relevant in the e-commerce domain, it is not quite in the movie domain. The relevant services communicates through gRPC network calls, gRPC being a high-performance framework which enables efficient communication between services, regardless of language[21], and is the network communication tool of choice in the TV 2 systems.

With these requirements in mind, the inference pipeline starts and ends with the client. The client makes a request to a master service `ai-api-blender` with details of which recommendation type, algorithm parameters such as how many items it want returned, but also more complex aspects such as the weights for our hybrid model w_1 and w_2. This service has been configured with connection informatoin to the different services it in turn needs, having these connections stored in memory allowing for blazingly fast communication. Amongst these connections we find `ai-api-item-filter` and `ai-api-blender-cf-next-sequence`, the relevant services for our inference. Recall

[21] https://grpc.io/

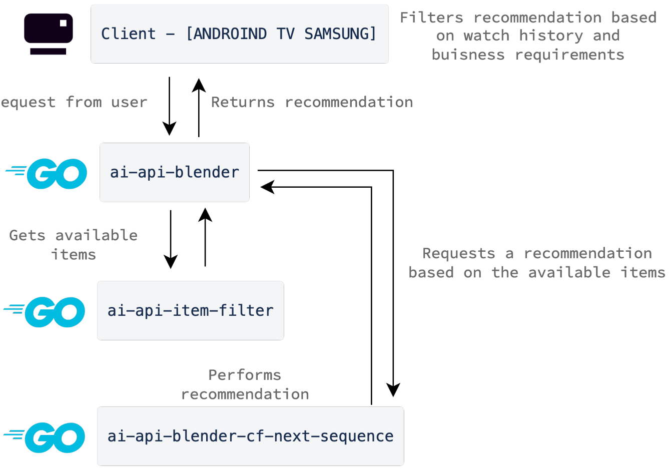

Client – [ANDROIND TV SAMSUNG]
Filters recommendation based on watch history and buisness requirements
Request from user
Returns recommendation
ai-api-blender
Gets available items
Requests a recommendation based on the available items
ai-api-item-filter
Performs recommendation
ai-api-blender-cf-next-sequence

Figure 3.8: The Inference Pipeline in more detail.

that our recommendation models are dependent on item ID's, needing to know which ID's it should score for. With this in mind `ai-api-blender` starts by requesting the `ai-api-item-filter` service for the list of allowed items, and then further requests a recommendation from the different algorithm services based on this item list, which in turn returns a scored set of times. Importantly however although the item filter service does filter out based on content, the users watch history is not accounted for there, rather being handled by the client itself through a separate API call to a service outside of this pipeline. This is an important factor as we need to recommend an excess amount of items back to the client, in case the user has already seen some of the recommendations. The CF model does not account for previously watched content, as we are scoring for factors that the users like based on the latent features, which in turn the items our user has watched will score high on. Handling content that the user has already seen is therefore important to avoid continuously recommending the same items that our user has already consumed.

Thanks to the modularity of the system we only needed to develop the algorithm service `ai-api-blender-cf-next-sequence` for this thesis, and registering the service in `ai-api-blender` to make it available to the larger recommender architecture. The objective of this service is to recommend an item based on the hybrid algorithm introduced in this thesis, meaning that for a user u, and the current item that user is watching j, return a list of items which we can present in the next-poster. To make a prediction, this application starts by storing the models into memory, it reads the relevant `ai-prod-nextsequence` and `ai-train-cf-batch` kafka topics, storing the rule scores from our Markov model into a hash map data structure referred to as a memory repository, so that we can simply lookup with a key j, and get the subsequent n scores from item j to any relevant item i. For the CF model we load the feature vectors into appropriate `mat.Dense` data types, allowing us to use the toolset available from the go mat package `gonum/mat`[22]. Through this we are

[22] https://pkg.go.dev/gonum.org/v1/gonum/mat

able to easily access the models, such as rule scores, or predict dot products for the users and items when applicable. The scoring itself happens at runtime, meaning that when the application receives a request from `ai-api-blender`, it scores the relevant items for our user through the function:

```
func (m *Model) calcUserItemsScores(ctx context.Context, row int) ([]float64, error) {
        rows, _ := m.UserFactors.Dims()
        if row > rows {
                return nil, errormsgs.ErrInternal
        }

        n := len(m.Items)
        scoresArr := make([]float64, n)
        scores := mat.NewVecDense(n, scoresArr)
        scores.MulVec(
                m.ItemFactors,
                m.UserFactors.RowView(row),
        )

        return scoresArr, nil
}
```

Without delving into the details of Go methodology, `MulVec` multiplies the dot-product as formulized as:

$$\hat{r}_{u,i} = q_i^T p_u \tag{3.21}$$

between the `ItemFactors` and `UserFactors`. Explaining the details of the Go code is out of scope for the thesis but note that by using a fast type safe language such as Go, computing the dot product for a single user u across the set of available items I is not a computational burden, as it results in a linear computational process, formally $O(n)$ scaling with the amount of items. Go is known for its speed and particularly in its capabilities of spinning up parallelized processed to handle requests Andrawos & Helmich (2017), meaning that the application can easily handle thousands of requests per second. When receiving a request we perform this calculations for all relevant items, we then get the scores from our Markov model by indexing the hash map containing the scores for item j to any relevant item i, and finally perform the hybrid recommendation based on the parameters in the request. This includes slicing the two recommendation sets to the subsequent n length, performing a softmax normalization on both, weighing the scores in both sets dependent on the w_1 and w_2 weights, and finally combining the scores to produce the final recommendation set. We then return the top k items from this recommendation. To account for items being removed client side, we always return 20 items on default, allowing the client to pick the top 3 items from this after performing its filtration's. The application will restart whenever there's a new available CF model on the kafka topic `ai-cf-model-info`, assuring that it always serves the most recent recommendations available. When restarting, the model also consumes the latest rules from `ai-prod-nextsequence`, assuring that this service is continuously up to date.

Overview

Thanks to this architectural implementation, the entire system finishes a training cycle for over two million user profiles in under an hour, including the process of aggregating and filtering the newest data. As we are continuously training, the model will only ever have one day delay, dealing with user feedback and new items each subsequent day.

3.4.3 Summary

Recall that the goals are to design a hybrid model, evaluating the performance of said hybrid model in an offline environment, then implementing and testing the model in a real-world environment, analyzing user engagement and implied satisfaction. Finally I also wish to compare the offline and online evaluation to explore the differences between these two evaluation methods. Designing the hybrid model is well described in Section 3.2.3, finding the necessary parameters and conducting the relevant performance benchmarks through the offline evaluation in Section 3.3.1. I then describe how the model can be evaluated in a real-world scenario in Section 3.3.2, explaining the deployment of such a model in Section 3.4.2.

These objectives are important to be able to answer our research questions:

- **RQ1**: To what extent does a Hybrid model outperform the traditional Collaborative Filtering model in regards to user engagement and click success?
- **RQ2**: To what extent can the outcome of an online evaluation be predicted based on an offline evaluation when utilizing a Sequence Aware approach in the Movie domain?

For **RQ1** I analyse the results of the offline and online evaluation scheme through the set metrics MRR, CTR, and click success metrics, comparing the Hyrbid model to the baseline *CF* model. For **RQ2** I compare the differences between the online and offline evaluation schemes to judge if the findings in the offline evaluation can simulate those in the online findings.

Chapter 4

Results

4.1 RQ 1: Performance of the Hybrid model

4.1.1 The Offline Evaluation

In regards to the offline evaluation the Hybrid model outperforms the *CF* model based on the MRR and CTR metrics, but at a much higher popularity score, alongside a lower coverage score. Although as seen in Table 4.1, the base *MC* Model outperforms the Hybrid model on both MRR and CTR with the exception of $k = 20$, where it marginally outperforms the *MC* model. This comes at a slight gain in the popularity scores however, showing the connection between performance and popularity. In terms of Coverage, as mentioned the *CF* model performs best but as the list length increases, the other models start performing better, with *MC* performing better than the Hybrid model. The *CF* model performs extremely poorly in this test scenario, which again reflects in much lower scores on the popularity metric. The best three scored configurations for each model, disregarding list length can be seen in Table A.6, and the best scoring for the online experiment case of $k = 3$ can be seen in Table A.8 in the Appendix.

Considering the normalization method for the Hybrid model, there are no performance differences between utilizing Log2 normalization between the top scoring configurations when ordering by CTR, however when ordering by MRR there is a clear indication that Log2 normalized scores come out slightly ahead (see Figure A.4 and Figure A.5. The same patterns appears when restricting k as well). In terms of which parameters lead to the best hybrid model, the parameter w_2, ergo the weight for the *MC* model, has lead to a high MRR and CTR, but also a subsequent high score on the popularity metrics. Finally, the *days* parameter is paramount to offline performance for both the Hybrid and *MC* model, for the configurations where $days = 30$ there is a substantial improvement over the same configuration with $days = 180$, see Table A.7 for the full overview. This improvement in accuracy does not come at the cost of popularity however, as the models trained on 180 days of data score much higher on the popularity metrics compared to their 30-day counterparts. Further, there is a clear indication that the lower k gets, CTR score drops significantly. Interestingly, whilst MRR also declines, it does so at a much lower rate, dropping from

Table 4.1: Best scoring from Offline Evaluation, sorted on MRR.

Model	k	MRR	CTR	GDPS	GAPS	APS	COV
H	@1	0.1473	0.1473	0.4728	**0.4362**	0.4625	0.1400
CF	@1	0.0028	0.0028	0.0821	0.0845	0.0802	**0.2240**
MC	@1	**0.1597**	**0.1597**	**0.4778**	0.4330	**0.4694**	0.1763
H	@3	0.2019	0.2739	0.4079	0.4400	0.4033	0.2240
CF	@3	0.0049	0.0080	0.0831	0.0868	0.0821	**0.3061**
MC	@3	**0.2152**	**0.2879**	**0.4196**	**0.4497**	**0.4169**	0.2896
H	@5	0.2200	0.3527	0.3534	0.3999	0.3479	0.2643
CF	@5	0.0060	0.0128	0.0828	0.0891	0.0820	**0.3508**
MC	@5	**0.2325**	**0.3633**	**0.3599**	**0.4083**	**0.3550**	0.3432
H	@10	0.2315	0.4391	**0.2684**	**0.3109**	0.2676	0.3166
CF	@10	0.0073	0.0221	0.0805	0.0930	0.0788	**0.4146**
MC	@10	**0.2428**	**0.4412**	0.2680	0.3081	**0.2684**	0.3977
H	@20	0.2396	**0.5556**	**0.2032**	**0.2152**	**0.2083**	0.3713
CF	@20	0.0082	0.0363	0.0745	0.0929	0.0734	**0.4784**
MC	@20	**0.2505**	0.5524	0.2025	0.2144	0.2077	0.4353

Best scoring configuration of each model H, CF, MC at different lenghts k, ordered by MRR. For GDPS, GAPS, APS and COV highest are marked but lower is better, else higher is better.

25% at $k = 20$ to 20% at $k = 3$, compared to the CTR decline of 55% to 27%. Meanwhile, the lower k gets, the higher the popularity scores are, showing a clear pattern that both the *MC* and Hybrid model tend to recommend popular content at higher positions.

4.1.2 Online Evaluation

The online evaluation of the best-performing Hybrid configuration shows that the Hybrid model does not outperform the baseline *CF* model in terms of CTR or MRR, but does outperform the *CF* model in terms of click success. As can be seen in Table 4.2, the popular baseline performs best in terms of CTR, followed by the *CF* model. In terms of the click success metrics however, the Hybrid model outperforms both *CF* and the popularity model *P*, whereas the *CF* performs best in terms of MRR. In relation to the popularity metrics, disregarding the obvious popular baseline, the Hybrid model overall recommends more popular content compared to the *CF* model. As for coverage, *CF* recommends the most diverse content, in total recommending 10% of the available catalogue at TV 2 Play after running for 10 days.

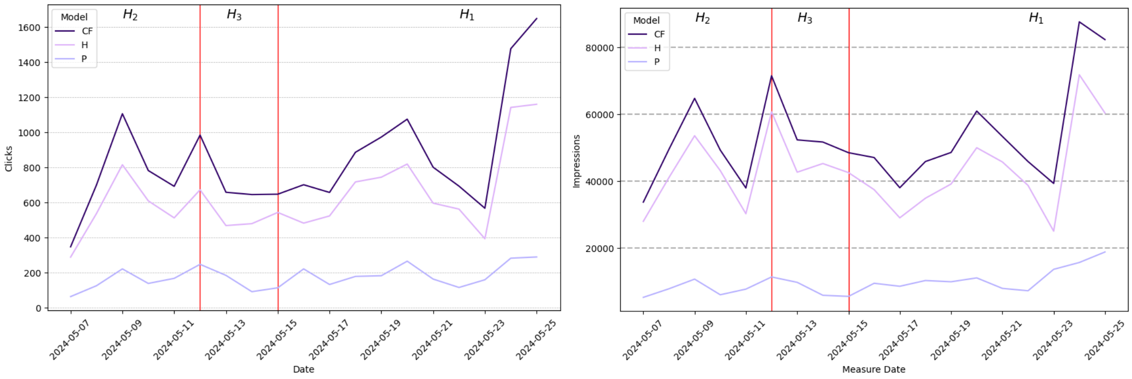
Table 4.2: Overview of results from the online experiment.

Model	CTR %	v@1min	v@2min	v@3min	v@50%	MRR	GAPS	GDPS	APS	COV
CF	0.0173	0.3554	0.3249	0.3063	0.2421	**0.0086**	0.2042	0.1712	0.1795	**0.1070**
H	0.0165	**0.3837**	**0.3564**	**0.3364**	**0.2669**	0.0083	0.2204	0.1863	0.2040	0.0908
P	**0.0178**	0.2910	0.2603	0.2462	0.1918	0.0076	**0.2934**	**0.2862**	**0.2912**	0.0188

Results from the Online Evaluation based on the best scoring configuration of H alongside the baseline CF and the fall-back popularity baseline P. Highest is highlighted, for the metrics GAPS, GDPS, APS lower is better, for the other metrics higher is better. Total impressions across the models are $\approx$ 1000000

4.2 RQ 2: Can we predict the Online results?

Based on the comparison of offline and online evaluation results for the three Hybrid configurations, I can identify the configuration that performs best in offline evaluation as also being the best performer compared to the baselines. However, it's not possible to predict the relative performance among the configurations. Given the comparison shown in Table 4.3, there are no clear indications that the online and offline evaluations of the Hybrid configurations lead to the same results in terms of scores. The offline evaluation reports much higher scores for CTR and MRR compared to the online counterparts, this is with the exception of the CTR scores of the configuration H_2, which scored close to one another. In terms of popularity metrics, the values are generally reported to be much higher in the offline evaluation compared to the online. Again with the exception of H_2, where the values are in turn reported much lower than that of the online counterpart, with the exception of Coverage.



(a) Clicks per Day

(b) Impressions per Day

Figure 4.1: Clicks and Impression variance for over the span of the experiment, red lines indicate the shift from which configuration was live at that time.

Although based on this table it looks like H_2 has the highest scores on all metrics, there was a lot of variance in the performance for all models across the different experiments (see Figure 4.2). This was expected for the Hybrid model as the parameters are changed during this time, but not for the *CF* and *P* models. This variance does not directly correlate with the changing impressions across the experiments as seen in Figure **??**, indicating that the variance of CTR is not directly connected to the impression numbers. Further, there is a large drop in the click success metrics for all models during the evaluation of configuration H_1, as seen in Figure 4.3, which in turn did not correlate to the score of the popularity metrics, as seen by the GAPS metric in the same graph. This drop in performance was isolated to the time span of the H_1 configuration, and not something eminent in the other configurations, see the Appendix for more detailed tables covering the breadth of the experiment. Importantly here is that although the configuration H_1 did not have the highest scores compared to the other configurations at face-value, it had the largest relative score in comparison to the *CF* and *P* baselines, see Table A.9 in the Appendix for the full overview.

Table 4.3: Offline versus Online evaluations.

Model	CTR %	v@1min	v@2min	v@3min	v@50%	MRR	GAPS	GDPS	APS	COV
H_1	$[n = 20, w_1 = 0.1, w_2 = 0.9, days = 30, normalization = log_2]$									
online	0.0165	0.3837	0.3564	0.3364	0.2669	0.0083	0.2204	0.1863	0.2040	0.0908
offline	0.2019	-	-	-	-	0.2739	0.4079	0.4400	0.4033	0.2240
H_2	$[n = 100, w_1 = 0.5, w_2 = 0.5, days = 180, normalization = log_2]$									
online	0.0134	0.5121	0.4777	0.4468	0.3670	0.0071	0.2740	0.2504	0.2805	0.0494
offline	0.0182	-	-	-	-	0.0275	0.1869	0.1924	0.1848	0.3392
H_3	$[n = 100, w_1 = 0.1, w_2 = 0.9, days = 180, normalization = log_2]$									
online	0.0114	0.4725	0.4329	0.4027	0.3208	0.0059	0.2559	0.1903	0.2343	0.0427
offline	0.1396	-	-	-	-	0.1951	0.4147	0.4358	0.4078	0.2615
CF										
online	0.0159	0.4107	0.3787	0.3586	0.2830	0.0081	0.2141	0.1779	0.1899	0.1092
offline	0.0049	-	-	-	-	0.0080	0.0831	0.0868	0.0821	0.3061

Comparison of offline and online evaluations of our three online cases, as well as the base CF model. For the metrics GAPS, GDPS, APS lower is better, for the other metrics higher is better. Recall that the *days* parameter entails how many days of data we train the Markov based models on. H_1 was live for 10 days, H_2 was live for 6 days, and finally H_3 was live for 3 days. The experiment had approx 100 000 impressions per day.

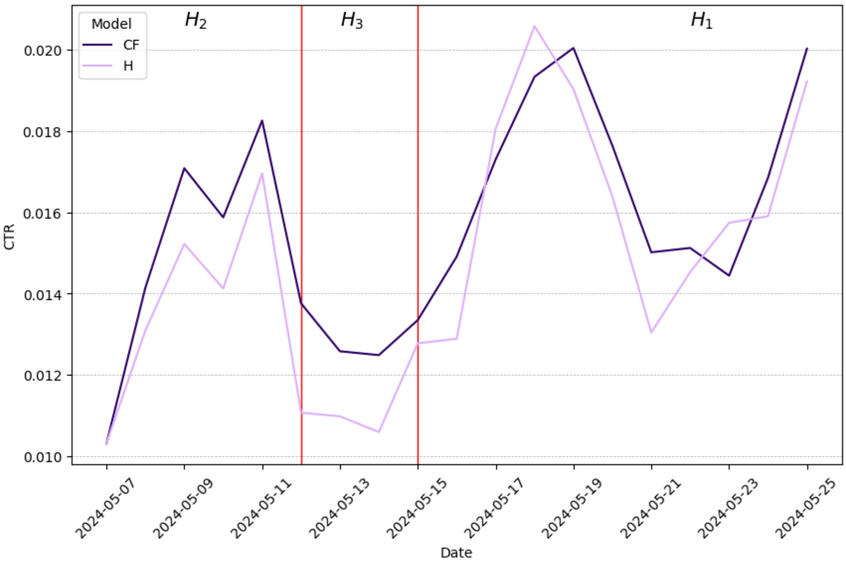


Figure 4.2: CTR for each model over the span of the experiment, red lines indicate the different cases

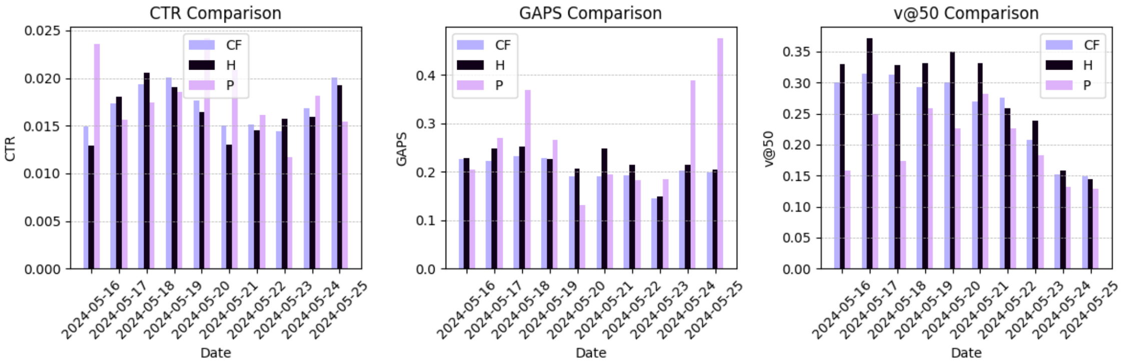


Figure 4.3: CTR, GAPS and v@50% for H_1, indicating the drop in the metric v@50%, which is not connected to the CTR or popularity metric GAPS dropping.

Chapter 5

Discussion

In this thesis, I have explored a solution to the next-poster recommendation problem. I set out with the objective of designing a Hybrid model, exploring said model in an offline evaluation, and further implementing and evaluating it in a real-world environment. Finally, I aimed to compare these results to each other to see if I could replicate results from a live environment in a simulated offline evaluation. In this chapter, I will first discuss the summary of the key findings, and then explore the research questions in relation to these. Then I will discuss the implications of the parameters and the different evaluation strategies, and finally explore the practical and theoretical implications of the methods.

When considering the online evaluation, the popular baseline model *P* consistently demonstrates the highest CTR, outperforming other models in this metric. The *CF* model follows closely, showcasing its effectiveness in driving user engagement. However, in terms of the click success metrics e.g. the implied user preference, the Hybrid model outperforms the other models in most cases. This suggests that the Hybrid model is better at aligning recommendations with user preferences and enhancing the overall user experience. This misalignment of engagement to click performance is interesting, it shows that although users are interested in the items presented by the popular model, they are more likely to keep watching the recommendations trying to adapt the context of the user. In other words, our popular and *CF* model are more prone to recommend click-bait content, which is not interesting for the user. This follows the arguments of Jannach & Zanker (2022), stating that only evaluating and optimizing for CTR can lead to a lower user satisfaction overall, and whilst I haven't obtained a clear explicit statement of user satisfaction as is usually found through user surveys, the user consuming the content they clicked on is arguably an indication of satisfaction. Note that I only make this assumption based on the time invested by our users, listening 50% through a song would not be considered an indication of a satisfied user, although spending up towards 30 minutes is arguably so. Metrics such as these however are arguably dependent on the user actively engaging with the content, if the user is guided to this content through e.g. an auto-play feature, a user watching 50% would not weigh as heavily as if they *actively chose what to watch*, simply because the lack of action, arguably does not weight the same as taking action.

Interestingly however is that across the models, even though the *CF* model outperforms the Hybrid model in terms of CTR, the Hybrid model has a tendency to recommend popular content, this might be due to how heavily the scores originating from the *MC* model are weighed, as there was heavy popularity tendency in the offline evaluation approach for the *MC* model. There is an argument here for the Hybrid model, whilst recommends highly popular content, being able to shift through this and present the most relevant popular content based on the users preferences. In support of this, the *CF* model performs better than the popular model in terms of the click success metrics, indicating that a perceived user's preferences does align more with the implied satisfaction they find with the recommendation. The *CF* model outperforms every configuration of the Hybrid model in terms of coverage, however this metric is rather skewed and not an indication of the whole truth due to the difference in impression counts. Although based on the offline results, the CF outperforms the Hybrid model when provided with the same exact impression counts, at all list lengths @k. The same is true for the gap between the popular and the Hybrid model, where *P*, acting as a fall-back mechanism, is exposed to vastly fewer impressions. Finally, although the measurements can't be directly compared across the different cases due to the vast shift in performance across time, comparatively to the other models the most performing offline Hybrid configuration does also outperform the other two configurations on all aspects in regards to the online evaluation.

In terms of the offline evaluations, some key findings are apparent. The base *MC* model outperforms the other models, but at the cost of very high popularity scores, this echoes the findings in Ludewig & Jannach (2018). Accordingly, the *CF* model does not perform well in this kind of evaluation, which in itself is no shock, however this does implicate the Hybrid model significantly, where a high score is more or less dependent on the w_2 parameter, ergo how high the items recommended from the *MC* model are scored. Recency matters in the evaluation scheme, training the *MC* model on 30 days rather than 180 shows higher scores across the board. Interestingly, when training the *MC* model on 180 days of data, the popularity scores increase significantly compared to the 30-day configuration, indicating that the *MC* model is prone to picking up popularity biases over a longer period of time, this can be mitigated by limiting the training to 30 days, at a surprisingly small coverage cost. The *MC* model, and therefore in turn the Hybrid model, manages to keep a (relatively) high MRR even though CTR tanks when k is reduced. This comes at the cost of recommending primarily popular content, yet again indicating the popularity aspect of the *MC* model. Due to the normalization strategy, setting $n = 20$ will result in more items appearing in the top-k from the *MC* model, in turn resulting in a higher score.

Finally, when comparing the offline and online results in regards to the next-poster problem, the best configuration from the offline evaluation does perform best compared to the alternative configurations. However it's not possible to directly predict the performance of a model, (e.g. the *CF* model) through this offline evaluation scheme. This echoes previous findings (Garcin et al., 2014; Beel & Langer, 2015; Maksai et al., 2015; Rossetti et al., 2016).

With these findings in mind, I can answer the research questions:

- **RQ1:** *To what extent does a Hybrid model outperform the traditional Collaborative Filtering model in regards to user engagement and click success?* The Hybrid model performs worse than a Collaborative Filtering approach in terms of user engagement

(CTR), but proves to be a better option when considering the clicks successes, at a slightly lover coverage of the catalogue.

- **RQ2:** *To what extent can the outcome of an online evaluation be predicted based on an offline evaluation when utilizing a Sequence Aware approach in the Movie domain?* When considering a sequence aware approach such as our hybrid model, we can identify the configuration that performs best in offline evaluation, as also being the best performer compared to the baselines, but we cannot predict the relative performance amongst different configurations.

There are many implications to be drawn from this result. First, the high popularity aspect of the *MC* model, and implied popularity aspect of the Hybrid model from the online evaluation. As with all models, this might be due to the inherent source of the data, the minimum or maximum support value for the rules, or finally the users' patterns. Internally TV 2 Play reports that the rows at the top of the site have the most clicks, a well-known aspect of most list-centered interfaces (Elahi et al., 2022). When looking at the landing page of the TV 2 Play platform, see Figure 5.1, the top row, ergo the most active, is the current top 10 items on the platform. Followed by a list of live-streamed channels, and finally the "Pick up where you left off" overview. Considering the user of Contiguous Sequential Patterns, the model inherently capture a lot of this first landing page, and therefore might simply end up following the editor-controlled frontage, recommending the most popular picks from these. A way around this could be to explore the options of non-sequential patters, or mining the sessions in such a way that the model captures more relevant events such as a search indication, or even putting a lower threshold on our minimum support value, indicating that the user had been active on the homepage for some time before choosing the next item to watch.

A further interesting observation in regards to popularity is that in the online evaluation of the second configuration, H_2, one would expect a lower popularity score due to weighing the presumed biased *MC* model less. This however ends up being the most popularity-centric results of all three cases.

The implications of RQ1 indicate that introducing a rather simple Sequence Aware model to solve the next-poster problem does increase the overall implied user satisfaction as the users are more likely to watch the provided recommendation, and whilst one might think that using a Sequence Aware model for a task which is inherently a sequence problem is an obvious reaction, there are very little research on the implementations of these models in the Movie domain. The domain matters, and most of the research on these models considers shorter events, or cases where there are little information on the content or users themselves. Adhering to user preferences matters, as the *CF* model did perform much better than expected in the online evaluation. It scoring as well as it did indicates that the evaluation methods utilized might not be directly applicable between online and offline evaluation for the Movie domain. In terms of RQ2 when trying to solve for the next-poster recommendation problem, utilizing an offline evaluation strategy can be beneficial to find the best parameters for a model, but ultimately the model should be evaluated through an online evaluation alongside a different configuration of the same model as a baseline, to ensure that the different parameters contribute to performance of the model as expected.

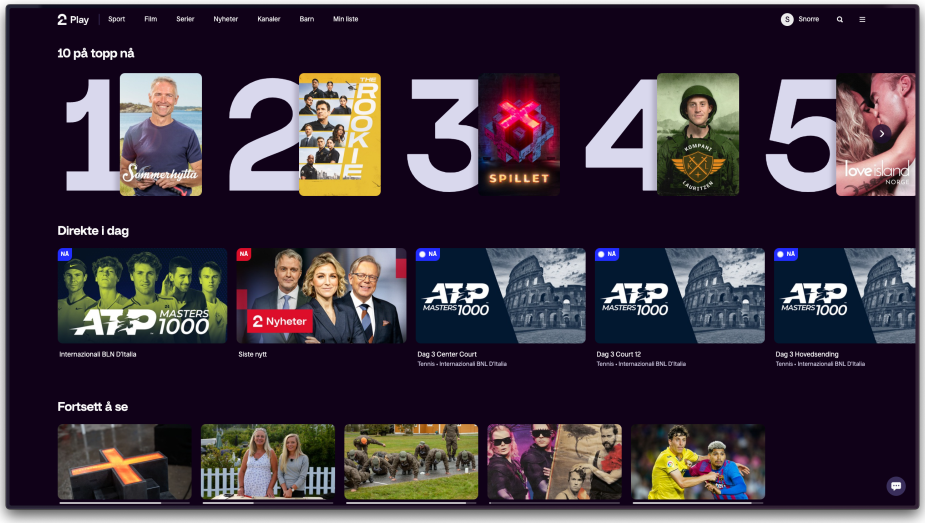


Figure 5.1: The front page of TV 2 Play at the time of writing.

In terms of the parameters, recency mattered a lot. The same two configurations of the Hybrid model scored vastly different, where the model trained on 30 days performed the best. This indicates that although there might be a bias towards popular content, considering the recent popular content scores is better than considering what was popular 100 days ago. Popularity has been shown to be a driver in terms of performance, as was showed in the News domain by Ludmann (2017), however as seen with the limited popularity model P, when popular content was presented to the user the overall click success dropped, whilst CTR scored high. This final point also brings forth the importance of other evaluation metrics beyond CTR, such as the viewing time metrics employed in this thesis.

CF scoring as well as it did might be an indication that for a presumed Sequence Aware problem, the solution might lie in user preferences. The *CF* model was not optimized for the problem, simply being employed as a baseline to compare our result with, however, if the model had been optimized in regards to the task through a new round of hyper-parameter optimization or grid-search, the results could have been different. There are some aspects of the CF model which could need some improvement, such as the BM25 weighing and how the max duration is limited to a flat value of 5000 seconds. Currently if a user has watched three episodes of any show in the last three years, it has the highest score in the user-item matrix possible, indicating a strong preference. However whilst watching a movie from start to end is an indication of preference, watching three episodes of a series is not necessarily an indication of preference, only watching three could even be an indication of dislike. In the current setting, these would be valued equally. Further considering a time-aware approach could be beneficial, not only do the results show that recency matters for a streaming platform, but many approaches considering the temporal aspect have shown an increase in accuracy over the years (Koren et al., 2009; Bogina et al., 2023). Considering that the k_1 parameter for the BM25, a user having watched a single episode of a show will result in that show scoring extremely low, as a high k_1 will put more emphasis on items scored high, and due to the roof of 5000, a lot of items will be scored high, resulting in a rich get richer scenario.

Another aspect is the Log_2 normalization of the Hybrid model, we saw that through the offline evaluation that normalizing the rule frequencies did lead to a higher MRR, but not a higher CTR, indicating that whilst the shift did not increase the predicted outcomes, it was able to move the relevant items higher up the rank ladder, which does matter a lot at shorter list lengths.

The final aspect is the deployment of the models. Whilst deploying a recommender model to the TV 2 System does not necessarily translate to deploying into another, having a cloud base micro service architecture did allow for a simple integration, and by training through different services the system can implement new items and users on a 1-day delay. Loading models into memory as I did in my service however might not be beneficial, as it can lead to a large memory consumption. With this in mind, I believe it might be better to separate the services completely, having one service for the *MC* model and one for the *CF* model (which already exists within TV 2), and then have a separate service responsible for weighing and mixing the items returned by the model-based services.

Chapter 6

Limitations, Future Work and Conclusion

There are some limitations to this work, most of them related to the online evaluation. Firstly, the short time span I evaluated our models on most likely had an impact on the results. As was seen in the results, not only did the measured scores vary greatly from day to day, but the fall in all measurement scores across models at the end of the experiment of the first configuration H_1 are hard to evaluate without a larger time span where such patterns might be noticed. Further the popularity model P applied as a fall-back did provide us with some indications of popularity issues in the Hybrid model, but for a true apples-to-apples comparison the model should have been used as a primary baseline as well, exposing it to more impressions and enabling me to either credit or discredit the large correlation between popularity and the Hybrid model. Finally in terms of the online evaluation, working in a live system comes with its own issues. When conducting the online experiment, a parallel experiment took place to explore the result of an auto-play feature for the next-poster problem, whilst the A/B groups were separate and I was able to remove the auto play "clicks" from the experiment, I was not able to remove the empty impressions originating from this experiment, skewing the overall CTR scores. Whilst this did not impact the relative performance of the models, it does impact the inter online-offline evaluations in terms of pure scores. On this same note how I conducted the offline experiment also affected this comparison, as although hit-rate and CTR are mathematically the same in my case, I did not include the instances where a user didn't pick an item, ergo empty impressions for the offline experiment, entailing that the sum under the denominator is magnitudes larger for CTR than it is HR. A further limitation is the lack of more Sequence Aware models in the offline evaluation, as noted in literature the simple models can hint at which more complex and in turn more performant model might be applicable, and by only considering one there is not enough grounds to be able to make an educated choice of which complex method to explore. This also entails that whilst our hybrid model did outperform the *CF* baseline, I cannot say for certain if this is due to the high performance of the *CF* model in the next-poster problem, or the bad performance of the Hybrid approach.

This thesis opens up several potential directions for future research. The next-poster problem is arguably a new problem in academia for the Movie domain. Whilst next-item recommendation models are well established in other domains, the ultimate goal of many of these models is to adapt the user's context, and the user's context is without a doubt dominated by *what* they are consuming or what their goals are. As argued in related work, how the MovieLens dataset has been used to evaluate the models is based on abstractions and assumptions, however, this does not disregard its performance. A possible future direction would therefore be to implement some of the more modern models evaluated on the MovieLens data such as GRU4REC Hidasi et al. (2015), HGRU4REC Quadrana et al. (2017), Caser Tang & Wang (2018) and AttRec Zhang et al. (2018) on our data alongside the *MC* model to see if they are able to outperform the *MC* model greatly, and if so, explore such an model in a real-world experiment.

In this thesis, I showed that employing a Hybrid approach to the next-poster problem gave an increase in implied user satisfaction by combining the broad user preferences from an ALS Collaborative Filtering approach with the Context Adaptation from a Markov Chain Sequence Aware approach. Through the cooperation with TV 2 Play, I was able to see that finding the best configuration of the Hybrid model by simulating the performance of the model through offline evaluation, does translate to finding the best performing model in an Online evaluation. Through said cooperation I am also able to show how such a model is implemented in a real-life system, together with the caveats and implications that entails, showing that through a cloud-based architecture based around micro services, its possible to retrain models dynamical, dealing with the influx of new data and new content apparent in any live system.

Appendix A

Appendix

A.1 Tables

Table A.1: Full online results of case one

	CTR %		**MRR**		**v@1min**		**v@2min**		**v@3min**		**v@50%**	
Model	CF	H	CF	H	CF	H	CF	H	CF	H	CF	H
2024-05-16	**0.0149**	0.0129	**0.0073**	0.0065	0.4379	**0.4523**	0.3966	**0.4336**	0.3823	**0.4108**	0.2996	**0.3299**
2024-05-17	0.0173	**0.0181**	0.0082	**0.0089**	0.4475	**0.5163**	0.4140	**0.4895**	0.3973	**0.4704**	0.3135	**0.3709**
2024-05-18	0.0193	**0.0206**	0.0091	**0.0100**	0.4492	**0.4714**	0.4097	**0.4324**	0.3837	**0.4059**	0.3126	**0.3278**
2024-05-19	**0.0200**	0.0190	**0.0100**	0.0094	0.4522	**0.4530**	0.3988	**0.4220**	0.3679	**0.4032**	0.2929	**0.3320**
2024-05-20	**0.0176**	0.0164	**0.0092**	0.0084	0.4447	**0.4969**	0.4047	**0.4640**	0.3814	**0.4444**	0.3005	**0.3492**
2024-05-21	**0.0150**	0.0130	**0.0076**	0.0068	0.4032	**0.4748**	0.3695	**0.4295**	0.3433	**0.4128**	0.2697	**0.3305**
2024-05-22	**0.0151**	0.0145	**0.0076**	0.0071	0.3968	**0.4181**	0.3608	**0.3879**	0.3463	**0.3541**	0.2756	0.2580
2024-05-23	0.0144	**0.0157**	0.0086	**0.0092**	0.3086	**0.3461**	0.2840	**0.3181**	0.2751	**0.3003**	0.2081	**0.2392**
2024-05-24	**0.0169**	0.0159	**0.0084**	0.0079	0.2214	**0.2277**	0.2045	**0.2093**	0.1889	**0.1961**	0.1517	**0.1576**
2024-05-25	**0.0200**	0.0192	**0.0097**	0.0093	0.2135	**0.2198**	0.2032	**0.2043**	0.1916	0.1853	**0.1486**	0.1448

	GAPS		**GDPS**		**APS**		**COV**					
Model	CF	H	CF	H	CF	H	CF	H				
2024-05-16	0.2265	**0.2292**	**0.1706**	0.1692	0.2002	**0.2128**	**0.0567**	0.0349	-	-	-	-
2024-05-17	0.2233	**0.2472**	0.1863	**0.2146**	0.2018	**0.2453**	**0.0604**	0.0432	-	-	-	-
2024-05-18	0.2326	**0.2518**	0.2045	**0.2312**	0.2135	**0.2530**	**0.0675**	0.0472	-	-	-	-
2024-05-19	**0.2275**	0.2261	0.1996	**0.2181**	0.2064	**0.2404**	**0.0713**	0.0484	-	-	-	-
2024-05-20	0.1898	**0.2068**	0.1545	**0.1705**	0.1679	**0.1983**	**0.0722**	0.0480	-	-	-	-
2024-05-21	0.1904	**0.2474**	0.1531	**0.1922**	0.1683	**0.2236**	**0.0674**	0.0466	-	-	-	-
2024-05-22	0.1927	**0.2135**	0.1543	**0.1702**	0.1688	**0.1987**	**0.0668**	0.0464	-	-	-	-
2024-05-23	0.1461	**0.1498**	0.1276	**0.1401**	0.1218	**0.1477**	**0.0516**	0.0335	-	-	-	-
2024-05-24	0.2022	**0.2135**	0.1763	**0.1810**	**0.1756**	0.1793	**0.0758**	0.0509	-	-	-	-
2024-05-25	0.1991	**0.2053**	0.1699	**0.1759**	0.1714	**0.1791**	**0.0741**	0.0593	-	-	-	-

Full online results of **Case 1**, with the relevant metrics, Note that for GAPS GDPS APS, lower is better, the higher is highlighted. Parameters: $n = 20$, $w_1 = 0.1$, $w_2 = 0.9$, $days = 30$, $normalization = log_2$

Table A.2: Full online results of case two

	CTR %		**MRR**		**v@1min**		**v@2min**		**v@3min**		**v@50%**	
Model	CF	H	CF	H	CF	H	CF	H	CF	H	CF	H
2024-05-07	0.0103	0.0103	0.0062	**0.0063**	0.5418	**0.5486**	**0.5014**	0.4896	**0.4697**	0.4688	0.3516	**0.3542**
2024-05-08	**0.0141**	0.0131	0.0073	0.0073	0.4671	**0.5084**	0.4286	**0.4711**	0.4171	**0.4339**	0.3143	**0.3371**
2024-05-09	**0.0171**	0.0152	**0.0090**	0.0078	0.4973	**0.5227**	0.4711	**0.4859**	0.4421	**0.4638**	0.3608	**0.3988**
2024-05-10	**0.0159**	0.0141	**0.0079**	0.0069	**0.5128**	0.4992	**0.4808**	0.4713	**0.4552**	0.4401	**0.3670**	0.3629
2024-05-11	**0.0183**	0.0170	**0.0090**	0.0082	**0.4827**	0.4922	0.4509	**0.4746**	0.4306	**0.4336**	0.3483	**0.3633**
2024-05-12	**0.0138**	0.0111	**0.0074**	0.0062	0.5091	**0.5134**	0.4654	**0.4762**	0.4411	**0.4435**	0.3486	**0.3646**

	GAPS		**GDPS**		**APS**		**COV**					
Model	CF	H	CF	H	CF	H	CF	H				
2024-05-07	0.1396	**0.3528**	0.1187	**0.2478**	0.1368	**0.2768**	**0.0048**	0.0025	-	-	-	-
2024-05-08	0.2521	**0.2981**	0.2077	**0.2437**	0.2311	**0.2848**	**0.0399**	0.0225	-	-	-	-
2024-05-09	0.2342	**0.2738**	0.1915	**0.2286**	0.2159	**0.2659**	**0.0478**	0.0307	-	-	-	-
2024-05-10	**0.2819**	0.2780	**0.2517**	0.2665	0.2707	**0.2901**	**0.0447**	0.0317	-	-	-	-
2024-05-11	0.2759	**0.2981**	0.2480	**0.2897**	0.2606	**0.3119**	**0.0457**	0.0300	-	-	-	-
2024-05-12	0.2277	**0.2373**	0.1913	**0.2337**	0.2019	**0.2615**	**0.0500**	0.0300	-	-	-	-

Full online results of **Case 2**, with the relevant metrics, Note that for GAPS GDPS APS, lower is better, the higher is highlighted. Parameters: $n = 100$, $w_1 = 0.5$, $w_2 = 0.5$, $days = 100$, $normalization = log_2$

A.2 Code

```
// .../NextFrequency.scala
// Read source data
val bridges = spark
  .read
  .format("parquet")
  .load(inpaths: _*)
  .filter(col("itemId") =!= col("nextItemId")) // remove binge
  .groupBy("itemId", "nextItemId")
  .sum("count")
  .withColumn("count", col("sum(count)"))


val windowSpecBridges = Window.partitionBy("itemId")
val bridgesWithFrequencyScore = bridges
  .withColumn("sumCount", sum(col("count")).over(windowSpecBridges))
  .withColumn("maxCount", max(col("count")).over(windowSpecBridges))
  .withColumn("minCount", min(col("count")).over(windowSpecBridges)) // Added for log normalization
  .withColumn("numItems", count("itemId").over(windowSpecBridges)) // this is for the rank score
  .filter(col("numItems") >= lit(bridgeThresholds)) // we clear any items with too few unique bridges
  .withColumn("frequencyScore", col("count") / col("sumCount"))
  // Prepare Log
  .withColumn("log2TransformedCount", log2(col("count") + 1))
  // +1 is for padding purposes and to help the 1-count bridges.
  .withColumn("log10TransformedCount", log10(col("count") + 1))
  .withColumn("minLog2Score", min("log2TransformedCount").over(windowSpecBridges))
  .withColumn("maxLog2Score", max("log2TransformedCount").over(windowSpecBridges))
  .withColumn("minLog10Score", min("log10TransformedCount").over(windowSpecBridges))
  .withColumn("maxLog10Score", max("log10TransformedCount").over(windowSpecBridges))

  // Linear normalization
  .withColumn("frequencyScoreNormalized", lit(minScore) + (col("count") / col("maxCount")) * (maxScore - minScore))
  // Log normalization
  .withColumn("frequencyScoreNormalizedLog2",
    (col("log2TransformedCount") - col("minLog2Score")) / (col("maxLog2Score") - col("minLog2Score")) * (lit(maxScor
  .withColumn("frequencyScoreNormalizedLog10",
    (col("log10TransformedCount") - col("minLog10Score")) / (col("maxLog10Score") - col("minLog10Score")) * (lit(max
```

Figure A.1: Code Snippet from the Spark Job `NextSequence.Scala`.

```
#implicit/nearest_neighbor.py

def bm25_weight(X, K1=100, B=0.8):
    """Weighs each row of a sparse matrix X by BM25 weighting"""
    # calculate idf per term (user)
    X = coo_matrix(X)

    N = float(X.shape[0])
    idf = log(N) - log1p(bincount(X.col))

    # calculate length_norm per document (artist)
    row_sums = np.ravel(X.sum(axis=1))
    average_length = row_sums.mean()
    length_norm = (1.0 - B) + B * row_sums / average_length

    # weight matrix rows by bm25
    X.data = X.data * (K1 + 1.0) / (K1 * length_norm[X.row] + X.data) * idf[X.col]
    return X
```

Figure A.2: BM25 Weighing from the Implicit Python Library

Table A.3: Full online results of case three

	CTR %		**MRR**		**v@1min**		**v@2min**		**v@3min**		**v@50%**	
Model	CF	H	CF	H	CF	H	CF	H	CF	H	CF	H
2024-05-13	**0.0126**	0.0110	**0.0067**	0.0058	0.4271	**0.4615**	0.3951	**0.4145**	0.3799	**0.3825**	0.2903	**0.3056**
2024-05-14	**0.0125**	0.0106	**0.0066**	0.0055	**0.5240**	0.4697	**0.4946**	0.4238	**0.4698**	0.3925	**0.3473**	0.3194
2024-05-15	**0.0133**	0.0128	**0.0071**	0.0064	0.4606	**0.4843**	0.4250	**0.4567**	0.4080	**0.4291**	**0.3369**	0.3352

	GAPS		**GDPS**		**APS**		**COV**					
Model	CF	H	CF	H	CF	H	CF	H				
2024-05-13	**0.2390**	0.2321	**0.1847**	0.1780	0.2121	**0.2213**	**0.0483**	0.0280	-	-	-	-
2024-05-14	0.2400	**0.2652**	0.1829	**0.1931**	0.2132	**0.2388**	**0.0522**	0.0297	-	-	-	-
2024-05-15	0.2337	**0.2640**	0.1764	**0.1961**	0.2073	**0.2393**	**0.0570**	0.0323	-	-	-	-

Full online results of **Case 3**, with the relevant metrics, Note that for GAPS GDPS APS, lower is better, the higher is highlighted. Parameters: $n = 100$, $w_1 = 0.1$, $w_2 = 0.9$, $days = 100$, $normalization = log_2$

Table A.4: Top Scoring from Offline Evaluation for model H, ordered by CTR

Model	Method	w_1	w_2	k	n	days	MRR	CTR	GDPS	GAPS	APS	COV
H	Count	0.1	0.9	20	20	30	0.2283	0.5556	0.2032	0.2152	0.2083	0.3591
H	Log2	0.1	0.9	20	20	30	0.2396	0.5556	0.2032	0.2152	0.2083	0.3713
H	Count	0.3	0.7	20	20	30	0.1866	0.5552	0.2035	0.2158	0.2083	0.3746
H	Log2	0.3	0.7	20	20	30	0.1939	0.5531	0.2032	0.2161	0.2078	0.3953
H	Log2	0.1	0.9	20	50	30	0.2341	0.5453	0.2022	0.2216	0.2057	0.3053
H	Log2	0.1	0.9	20	100	30	0.2280	0.5363	0.2021	0.2222	0.2055	0.2484
H	Log2	0.3	0.7	20	50	30	0.1366	0.5140	0.2007	0.2224	0.2025	0.3285
H	Count	0.1	0.9	20	50	30	0.1893	0.5132	0.2006	0.2225	0.2022	0.3127
H	Count	0.3	0.7	20	50	30	0.0974	0.5042	0.1994	0.2214	0.2010	0.3271
H	Count	0.1	0.9	20	20	180	0.1793	0.4988	0.2603	0.2726	0.2648	0.4737

Best 10 configuratinos H, ordered by CTR

Table A.5: Top Scoring from Offline Evaluation for model H, ordered by MRR

Model	Method	w_1	w_2	k	n	days	MRR	CTR	GDPS	GAPS	APS	COV
H	Log2	0.1	0.9	20	20	30	0.2396	0.5556	0.2032	0.2152	0.2083	0.3713
H	Log2	0.1	0.9	20	50	30	0.2341	0.5453	0.2022	0.2216	0.2057	0.3053
H	Log2	0.1	0.9	10	20	30	0.2315	0.4391	0.2684	0.3109	0.2676	0.3166
H	Count	0.1	0.9	20	20	30	0.2283	0.5556	0.2032	0.2152	0.2083	0.3591
H	Log2	0.1	0.9	20	100	30	0.2280	0.5363	0.2021	0.2222	0.2055	0.2484
H	Log2	0.1	0.9	10	50	30	0.2263	0.4323	0.2659	0.3099	0.2645	0.2648
H	Log2	0.1	0.9	10	100	30	0.2206	0.4281	0.2687	0.3116	0.2679	0.2015
H	Count	0.1	0.9	10	20	30	0.2200	0.4365	0.2677	0.3108	0.2665	0.3117
H	Log2	0.1	0.9	5	20	30	0.2200	0.3527	0.3534	0.3999	0.3479	0.2643
H	Log2	0.1	0.9	5	50	30	0.2146	0.3444	0.3505	0.3962	0.3450	0.2169

Best 10 configurations for each model H, ordered by MRR

Table A.6: Top Three Scoring from Offline Evaluation, sorted on MRR

Model	Method	w_1	w_2	k	n	days	MRR	CTR	GDPS	GAPS	APS	COV
H	Log2	0.1	0.9	20	20	30	0.2395	**0.5555**	0.2032	0.2151	0.2083	0.3713
H	Log2	0.1	0.9	20	50	30	0.2340	0.5452	0.2022	0.2216	0.2056	0.3052
H	Log2	0.1	0.9	10	20	30	0.2314	0.4390	0.2683	0.3109	0.2675	0.3166
CF	-	-	-	20	-	-	0.0082	0.0363	***0.0745**	0.0929	***0.0733**	**0.4784**
CF	-	-	-	10	-	-	0.0072	0.0221	0.0804	0.0929	0.0788	0.4145
CF	-	-	-	5	-	-	0.0060	0.0127	0.0828	***0.0890**	0.0819	0.3507
MC	-	-	-	20	-	30	**0.2505**	0.5524	0.2025	0.2144	0.2077	0.4353
MC	-	-	-	10	-	30	0.2428	0.4412	0.2680	0.3081	0.2684	0.3977
MC	-	-	-	5	-	30	0.2325	0.3633	0.3599	0.4083	0.3550	0.3432

Best three configurations for each model H, CF, MC, ordered by MRR. For * lower is better, else higher is better.

Table A.7: Top Three Scoring based on days

Model	Method	w_1	w_2	k	n	days	MRR	CTR	GDPS	GAPS	APS	COV
H	Log2	0.1	0.9	20	20	30	0.2396	0.5556	0.2032	0.2152	0.2083	0.3713
H	Log2	0.1	0.9	20	50	30	0.2341	0.5453	0.2022	0.2216	0.2057	0.3053
H	Log2	0.1	0.9	10	20	30	0.2315	0.4391	0.2684	0.3109	0.2676	0.3166
⋮			⋮				⋮			⋮		
H	Log2	0.1	0.9	3	20	30	0.2019	0.2739	0.4079	0.4400	0.4033	0.2240
⋮			⋮				⋮			⋮		
H	Log2	0.1	0.9	20	20	180	0.1931	0.4988	0.2603	0.2726	0.2648	0.4767
H	Log2	0.1	0.9	10	20	180	0.1849	0.3784	0.3199	0.3467	0.3164	0.4035
H	Log2	0.1	0.9	20	50	180	0.1832	0.4793	0.2518	0.2694	0.2535	0.3947
H	Log2	0.1	0.9	3	20	180	0.1576	0.2218	0.4308	0.4523	0.4244	0.3053
MC	-	-	-	20	-	30	0.2505	0.5524	0.2025	0.2144	0.2077	0.4353
MC	-	-	-	10	-	30	0.2428	0.4412	0.2680	0.3081	0.2684	0.3977
MC	-	-	-	5	-	30	0.2325	0.3633	0.3599	0.4083	0.3550	0.3432
⋮			⋮				⋮			⋮		
MC	-	-	-	3		30	0.2152	0.2879	0.4196	0.4497	0.4169	0.2896
⋮			⋮				⋮			⋮		
MC	-	-	-	20	-	180	0.2075	0.4970	0.2596	0.2720	0.2641	0.5419
MC	-	-	-	10	-	180	0.1997	0.3830	0.3214	0.3442	0.3185	0.4835
MC	-	-	-	5	-	180	0.1898	0.3077	0.3987	0.4296	0.3891	0.4067
MC	-	-	-	3		180	0.1753	0.2445	0.4492	0.4730	0.4451	0.3484

Best three configurations for each model H, MC, and the scores for the configrutation $k = 3$, ordered by MRR based on the Days parameter.

Table A.8: Top Scoring from Offline Evaluation where $n = 3$

Model	Method	w_1	w_2	k	n	days	MRR	CTR	GDPS	GAPS	APS	COV
H	Log2	0.1	0.9	3	20	30	0.2019	0.2739	0.4079	0.4400	0.4033	0.2240
H	Log2	0.1	0.9	3	50	30	0.1964	0.2653	0.4043	0.4383	0.3986	0.1795
H	Log2	0.1	0.9	3	100	30	0.1902	0.2580	0.4043	0.4369	0.3974	0.1212
CF	-	-	-	3	-	-	0.0049	0.0079	***0.0831**	***0.0868**	***0.0821**	0.3060
MC	-	-	-	3	-	30	**0.2152**	**0.2879**	0.4196	0.4497	0.4169	0.2896
MC	-	-	-	3	-	180	0.1753	0.2445	0.4492	0.4730	0.4451	**0.3484**

Best three configurations for each model *H*, *CF*, *MC* where $k = 3$, recall that this is the list lenght presented to our user in the online experiment, ordered by MRR, for * lower is better, else higher is better.

Table A.9: Full overview of results from online experiment in regards to all three model configurations

Model	CTR %	v@1min	v@2min	v@3min	v@50%	MRR	GAPS	GDPS	APS	COV
Configuration 1, *impressions* ≈ 1000000, running for 10 days										
CF	0.0173	0.3554	0.3249	0.3063	0.2421	**0.0086**	0.2042	0.1712	0.1795	**0.1070**
H	0.0165	**0.3837**	**0.3564**	**0.3364**	**0.2669**	0.0083	0.2204	0.1863	0.2040	0.0908
P	**0.0178**	0.2910	0.2603	0.2462	0.1918	0.0076	**0.2934**	**0.2862**	**0.2912**	0.0188
Configuration 2	*impressions* ≈ 600000, running for 6 days									
CF	0.0150	0.4990	0.4643	0.4407	0.3496	0.0079	0.2521	0.2158	0.2336	**0.0689**
H	0.0134	**0.5121**	**0.4777**	**0.4468**	**0.3670**	0.0071	0.2740	0.2504	0.2805	0.0494
P	**0.0199**	0.4662	0.4162	0.4058	0.3143	**0.0087**	**0.3263**	**0.3349**	**0.3320**	0.0075
Configuration 3	*impressions* ≈ 300000, running for 3 days									
CF	0.0128	0.4703	**0.4379**	**0.4190**	**0.3246**	0.0068	0.2373	0.1810	0.2106	**0.0694**
H	0.0114	**0.4725**	0.4329	0.4027	0.3208	0.0059	**0.2559**	0.1903	0.2343	0.0427
P	**0.0186**	0.4175	0.3686	0.3634	0.2938	**0.0076**	0.2316	**0.2564**	**0.2478**	0.0068

All cases with the models CF, H and the popularity baseline P, for the metrics GAPS, GDPS, APS lower is better, but the highest is highlighted, for the other metrics higher is better.

Bibliography

Adomavicius, G., Bauman, K., Tuzhilin, A., & Unger, M. (2022). Context-aware recommender systems: From foundations to recent developmentscontext-aware recommender systems. In F. Ricci, L. Rokach, & B. Shapira (Eds.), *Recommender Systems Handbook* (pp. 211–250). New York, NY: Springer US. URL: https://doi.org/10.1007/978-1-0716-2197-4_6. doi:10.1007/978-1-0716-2197-4_6.

Adomavicius, G., & Kwon, Y. (2011). Improving aggregate recommendation diversity using ranking-based techniques. *IEEE Transactions on Knowledge and Data Engineering*, *24*, 896–911.

Adomavicius, G., & Tuzhilin, A. (2011). Context-aware recommender systems. In F. Ricci, L. Rokach, B. Shapira, & P. B. Kantor (Eds.), *Recommender Systems Handbook* (pp. 217–253). Boston, MA: Springer US. URL: https://doi.org/10.1007/978-0-387-85820-3_7. doi:10.1007/978-0-387-85820-3_7.

Aggarwal, C. C. et al. (2016). *Recommender systems* volume 1. Springer.

Agrawal, R., Imieliński, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data* (pp. 207–216).

Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. In *Proceedings of the eleventh international conference on data engineering* (pp. 3–14). IEEE.

Agrawal, R., Srikant, R. et al. (1994). Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB* (pp. 487–499). Santiago volume 1215.

Aljunid, M. F., & Manjaiah, D. H. (2019). Movie recommender system based on collaborative filtering using apache spark. In V. E. Balas, N. Sharma, & A. Chakrabarti (Eds.), *Data Management, Analytics and Innovation* (pp. 283–295). Singapore: Springer Singapore.

Andrawos, M., & Helmich, M. (2017). *Cloud Native Programming with Golang: Develop microservice-based high performance web apps for the cloud with Go*. Packt Publishing Ltd.

Bambini, R., Cremonesi, P., & Turrin, R. (2011). A recommender system for an iptv service provider: a real large-scale production environment. In F. Ricci, L. Rokach, B. Shapira, & P. B. Kantor (Eds.), *Recommender Systems Handbook* (pp. 299–331). Boston, MA:

Springer US. URL: https://doi.org/10.1007/978-0-387-85820-3_9. doi:10.1007/978-0-387-85820-3_9.

Beel, J., & Langer, S. (2015). A comparison of offline evaluations, online evaluations, and user studies in the context of research-paper recommender systems. In *Research and Advanced Technology for Digital Libraries: 19th International Conference on Theory and Practice of Digital Libraries, TPDL 2015, Poznań, Poland, September 14-18, 2015, Proceedings 19* (pp. 153–168). Springer.

Bennett, J., Lanning, S. et al. (2007). The netflix prize. In *Proceedings of KDD cup and workshop* (p. 35). New York volume 2007.

Bernardis, C., Dacrema, M. F., Maurera, F. B. P., Quadrana, M., Scriminaci, M., & Cremonesi, P. (2022). From data analysis to intent-based recommendation: An industrial case study in the video domain. *IEEE Access*, *10*, 14779–14796. doi:10.1109/ACCESS.2022.3148434.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, *3*, 993–1022.

Bogina, V., Kuflik, T., Jannach, D., Bielikova, M., Kompan, M., & Trattner, C. (2023). Considering temporal aspects in recommender systems: a survey. *User Modeling and User-Adapted Interaction*, *33*, 81–119. URL: https://doi.org/10.1007/s11257-022-09335-w. doi:10.1007/s11257-022-09335-w.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, *12*, 331–370.

Coulouris, G. F., Dollimore, J., & Kindberg, T. (2005). *Distributed systems: concepts and design*. pearson education.

Covington, P., Adams, J., & Sargin, E. (2016). Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems* (pp. 191–198).

Elahi, M., Jannach, D., Skjærven, L., Knudsen, E., Sjøvaag, H., Tolonen, K., Holmstad, Ø., Pipkin, I., Throndsen, E., Stenbom, A., Fiskerud, E., Oesch, A., Vredenberg, L., & Trattner, C. (2022). Towards responsible media recommendation. *AI and Ethics*, *2*, 103–114. URL: https://doi.org/10.1007/s43681-021-00107-7. doi:10.1007/s43681-021-00107-7.

Fam Vivian Bekkengen (2024). Dette er de mest populære strømmetjenestene i norge. Retrieved from https://www.ssb.no/kultur-og-fritid/tids-og-mediebruk/statistikk/norsk-mediebarometer/artikler/dette-er-de-mest-populaere-strommetjenestene-i-norge.

Fan, Q., Jiao, L., Dai, C., Deng, Z., & Zhang, R. (2019). Golang-based poi discovery and recommendation in real time. In *2019 20th IEEE International Conference on Mobile Data Management (MDM)* (pp. 527–532). doi:10.1109/MDM.2019.00114.

Fernández-Tobías, I., Cantador, I., Kaminskas, M., & Ricci, F. (2012). Cross-domain recommender systems: A survey of the state of the art. In *Spanish conference on information retrieval*. sn volume 24.

Garcin, F., Faltings, B., Donatsch, O., Alazzawi, A., Bruttin, C., & Huber, A. (2014). Offline and online evaluation of news recommender systems at swissinfo. ch. In *Proceedings of the 8th ACM Conference on Recommender systems* (pp. 169–176).

Goldberg, K., Roeder, T., Gupta, D., & Perkins, C. (2001). Eigentaste: A constant time collaborative filtering algorithm. *information retrieval*, *4*, 133–151.

Han, P., Xie, B., Yang, F., & Shen, R. (2004). A scalable p2p recommender system based on distributed collaborative filtering. *Expert Systems with Applications*, *27*, 203–210. URL: https://www.sciencedirect.com/science/article/pii/S0957417404000065. doi:https://doi.org/10.1016/j.eswa.2004.01.003.

Harper, F. M., & Konstan, J. A. (2015). The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, *5*, 1–19.

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

Hidasi, B., Quadrana, M., Karatzoglou, A., & Tikk, D. (2016). Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *Proceedings of the 10th ACM conference on recommender systems* (pp. 241–248).

Hofmann, T. (2004). Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, *22*, 89–115.

Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining* (pp. 263–272). doi:10.1109/ICDM.2008.22.

Jannach, D., & Ludewig, M. (2017). When recurrent neural networks meet the neighborhood for session-based recommendation. In *Proceedings of the eleventh ACM conference on recommender systems* (pp. 306–310).

Jannach, D., & Zanker, M. (2022). Value and impact of recommender systems. In F. Ricci, L. Rokach, & B. Shapira (Eds.), *Recommender Systems Handbook* (pp. 519–546). New York, NY: Springer US. URL: https://doi.org/10.1007/978-1-0716-2197-4_14. doi:10.1007/978-1-0716-2197-4_14.

Jannach, D., Zanker, M., Felfernig, A., & Friedrich, G. (2010). *Recommender systems: an introduction*. Cambridge University Press.

Kamehkhosh, I., Jannach, D., & Ludewig, M. (2017). A comparison of frequent pattern techniques and a deep learning method for session-based recommendation. In *RecTemp@ RecSys* (pp. 50–56).

Koren, Y., & Bell, R. (2011). Advances in collaborative filtering. In F. Ricci, L. Rokach, B. Shapira, & P. B. Kantor (Eds.), *Recommender Systems Handbook* (pp. 145–186). Boston, MA: Springer US. URL: https://doi.org/10.1007/978-0-387-85820-3_5. doi:10.1007/978-0-387-85820-3_5.

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, *42*, 30–37.

Koren, Y., Rendle, S., & Bell, R. (2022). Advances in collaborative filtering. In F. Ricci, L. Rokach, & B. Shapira (Eds.), *Recommender Systems Handbook* (pp. 91–142). New York, NY: Springer US. URL: https://doi.org/10.1007/978-1-0716-2197-4_3. doi:10.1007/978-1-0716-2197-4_3.

Latifi, S., Mauro, N., & Jannach, D. (2021). Session-aware recommendation: A surprising quest for the state-of-the-art. *Information Sciences*, *573*, 291–315.

Little, R. J., & Rubin, D. B. (2019). *Statistical analysis with missing data* volume 793. John Wiley & Sons.

Liu, D. C., Rogers, S., Shiau, R., Kislyuk, D., Ma, K. C., Zhong, Z., Liu, J., & Jing, Y. (2017). Related pins at pinterest: The evolution of a real-world recommender system. In *Proceedings of the 26th international conference on world wide web companion* (pp. 583–592).

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

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

Maksai, A., Garcin, F., & Faltings, B. (2015). Predicting online performance of news recommender systems through richer evaluation metrics. In *Proceedings of the 9th ACM Conference on Recommender Systems* (pp. 179–186).

Mobasher, B., Dai, H., Luo, T., & Nakagawa, M. (2002). Using sequential and non-sequential patterns in predictive web usage mining tasks. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.* (pp. 669–672). IEEE.

Mobasher, B., & Nakagawa, M. (2002). Impact of site characteristics on recommendation models based on association rules and sequential patterns. In *IEEE International Conference on Data Mining (ICDM'2002), Maebashi City, Japan*. Citeseer.

Nasir, M., & Ezeife, C. I. (2023). A survey and taxonomy of sequential recommender systems for e-commerce product recommendation. *SN Computer Science*, *4*, 708.

Norris, J. R. (1998). *Markov chains*. 2. Cambridge university press.

Oard, D. W., Kim, J. et al. (1998). Implicit feedback for recommender systems. In *Proceedings of the AAAI workshop on recommender systems* (pp. 81–83). Madison, WI volume 83.

Quadrana, M., Cremonesi, P., & Jannach, D. (2018). Sequence-aware recommender systems. *ACM Comput. Surv.*, *51*. URL: https://doi.org/10.1145/3190616. doi:10.1145/3190616.

Quadrana, M., Karatzoglou, A., Hidasi, B., & Cremonesi, P. (2017). Personalizing session-based recommendations with hierarchical recurrent neural networks. In *proceedings of the Eleventh ACM Conference on Recommender Systems* (pp. 130–137).

Ricci, F., Rokach, L., & Shapira, B. (2011). Introduction to recommender systems handbook. In F. Ricci, L. Rokach, B. Shapira, & P. B. Kantor (Eds.), *Recommender Systems Handbook* (pp. 1–35). Boston, MA: Springer US. URL: https://doi.org/10.1007/978-0-387-85820-3_1. doi:10.1007/978-0-387-85820-3_1.

Robertson, S., Zaragoza, H., & Taylor, M. (2004). Simple bm25 extension to multiple weighted fields. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management* (pp. 42–49).

Robertson, S. E., & Sparck Jones, K. (1994). *Simple, proven approaches to text retrieval*. Technical Report University of Cambridge Computer Laboratory.

Robertson, S. E., Walker, S., Jones, S., Hancock-Beaulieu, M. M., & Gatford, M. (1995). Okapi at trec-3. In *NIST Special Publication 500-225: Overview of the Third Text REtrieval Conference (TREC-3)* (pp. 109–126).

Rossetti, M., Stella, F., & Zanker, M. (2016). Contrasting offline and online results when evaluating recommendation algorithms. In *Proceedings of the 10th ACM conference on recommender systems* (pp. 31–34).

Roy, D., & Dutta, M. (2022). A systematic review and research perspective on recommender systems. *Journal of Big Data*, *9*, 59. URL: https://doi.org/10.1186/s40537-022-00592-5. doi:10.1186/s40537-022-00592-5.

Salakhutdinov, R., Mnih, A., & Hinton, G. (2007). Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning* (pp. 791–798).

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web* (pp. 285–295).

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., & Dennison, D. (2015). Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, *28*.

SSB (2024). Norwegian mediebarometer: Statistics on media usage and access. Statistics Norway. Retrieved from https://www.ssb.no/kultur-og-fritid/tids-og-mediebruk/statistikk/norsk-mediebarometer.

Tang, J., & Wang, K. (2018). Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the eleventh ACM international conference on web search and data mining* (pp. 565–573).

Töscher, A., & Jahrer, M. (2009). The bigchaos solution to the netflix grand prize. *unknown*, .

Véras, D., Prota, T., Bispo, A., Prudêncio, R., & Ferraz, C. (2015). A literature review of recommender systems in the television domain. *Expert Systems with Applications*, *42*, 9046–9076.

Wang, J., Pouwelse, J., Fokker, J., de Vries, A. P., & Reinders, M. J. T. (2008). Personalization on a peer-to-peer television system. *Multimedia Tools and Applications*, *36*, 89–113. URL: https://doi.org/10.1007/s11042-006-0075-6. doi:10.1007/s11042-006-0075-6.

Wang, S., Cao, L., Wang, Y., Sheng, Q. Z., Orgun, M. A., & Lian, D. (2021). A survey on session-based recommender systems. *ACM Computing Surveys (CSUR)*, *54*, 1–38.

Yap, G.-E., Li, X.-L., & Yu, P. S. (2012). Effective next-items recommendation via personalized sequential pattern mining. In *Database Systems for Advanced Applications: 17th International Conference, DASFAA 2012, Busan, South Korea, April 15-19, 2012, Proceedings, Part II 17* (pp. 48–64). Springer.

Zeng, Z., Lin, J., Li, L., Pan, W., & Ming, Z. (2019). Next-item recommendation via collaborative filtering with bidirectional item similarity. *ACM Transactions on Information Systems (TOIS)*, *38*, 1–22.

Zhang, S., Tay, Y., Yao, L., & Sun, A. (2018). Next item recommendation with self-attention. *arXiv preprint arXiv:1808.06414*, .

Zhao, Z., Hong, L., Wei, L., Chen, J., Nath, A., Andrews, S., Kumthekar, A., Sathiamoorthy, M., Yi, X., & Chi, E. (2019). Recommending what video to watch next: a multitask ranking system. In *Proceedings of the 13th ACM Conference on Recommender Systems* (pp. 43–51).

Zheng, H., Wang, D., Zhang, Q., Li, H., & Yang, T. (2010). Do clicks measure recommendation relevancy? an empirical user study. In *Proceedings of the Fourth ACM Conference on Recommender Systems* RecSys '10 (p. 249–252). New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/1864708.1864759. doi:10.1145/1864708.1864759.