A STATE OF THE ART IN MACHINE LEARNING RECOMMENDATION SYSTEMS

A Thesis Research Paper

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the
School of Engineering and Applied Science
University of Virginia
In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science in Computer Science

Brendan Keaton

December 2023

Elizabeth Orrico, Department Department of Computer Science

A STATE OF THE ART IN MACHINE LEARNING RECOMMENDATION SYSTEMS

Brendan Keaton
Department of Computer Science
University of Virginia
BrendanRKeaton@gmail.com

ABSTRACT

One of the most exciting emerging technologies shaping the internet is recommendation algorithms. Recommendation systems take in countless types of data to recommend new items to users. This paper begins with an introduction and summarizes the base knowledge necessary to understand the world of recommendation systems. Following this, the paper will discuss current technologies for the reader to understand where the industry currently stands. The paper will then delve into the research conducted in the last few years, to project the future of the industry. Finally, the paper will conclude with what problems have arisen and need to be solved in the future.

1 INTRODUCTION

The exponential growth of online content in the early twenty-first century has made it increasingly difficult for consumers to find content that suits their interests. This has prompted developers and researchers alike to manufacture methods to keep users engaged with their platform and content above all other outlets. One of the leading solutions to keep users engaged has been through the use of recommendation systems.

Machine learning is used by nearly every large, online platform to curate posts for individual users, with some of the most successful examples being Netflix, Google, Twitter, and more recently, TikTok. According to Quinn (2022), machine learning has heavily contributed to the massive success of companies, such as TikTok, which uses data retrieved from its users to curate the perfect feed (para. 10). As discussed by Munawar (2017), the effectiveness and efficiency of these algorithms are becoming increasingly important to the success of businesses (p. 196). Due to this, research in the field of recommendation systems has become of prime interest to corporations and universities around the world. This explosion of research has produced thousands of research papers, books, and articles, many of which are extremely complex and not consumable by the general public. The goals of this state-of-the-art report are to (1) provide base knowledge to those not currently acquainted with the world of recommendation systems and machine learning, (2) give an in-depth description of the current algorithms and types of machine learning used in recommendation systems, (3) explore and subsequently discuss new methods being discovered and employed by researchers worldwide, and (4) theorize future advancements based on the needs of the industry.

2 BASE KNOWLEDGE

To dive into the world of recommendation systems, and the complex algorithms that come with them, there must first be a framework for understanding the topics. This section will aim to achieve a baseline comprehension of these topics for those unfamiliar with the field of machine learning and social media.

2.1 What are Recommendation Systems

Simply put, recommendation systems are algorithms to suggest the most relevant media, products, or information to the current user. As an example, if a user were to buy item X, is it more likely that they would then be interested in item Y or item Z? Recommendation systems are an automatic means for answering these kinds of questions quickly and accurately. These algorithms are extremely useful in modern-day applications as there can be millions of possible items shown to users, and offering these at random will mean that most items are of no interest to the user[1].

2.2 Why are Recommendation Systems Used

As explained in Ricci (2015), there are five main motivators for websites and applications to use recommendation systems; the first three of which will be focused on in this section. These reasons include (p. 5-6):

- To increase the number of items being sold
- To sell a wider variety of items
- To increase user satisfaction
- To increase user fidelity
- To have a better understanding of what the user wants

The first motivator for recommender systems, to increase the number of items being sold, is perhaps the most important. Ultimately, the goal of all companies using these systems is to make money; thus, the primary benefit to them is the increased sales rate. This point is equivalent to increasing user retention rates on social media apps, such as TikTok and Twitter. These applications make a large portion of their income through advertisements, meaning that the longer a user spends on the application, the more money the company will make.

The second motivator, to sell a wider variety of items, is adequately explained by Ricci (2015) when he gives the example that "... in a tourist recommender system, the service provider is interested in promoting all the places of interest in a tourist area, not just the most popular ones. This could be difficult without a recommender system since the service provider cannot afford the risk of advertising places that are not likely to suit a particular user's taste." (p. 5).

The third motivator, to increase user satisfaction aims to make the entire experience for the user better. In short, if the user is only being shown items they are interested in, they are much more likely to enjoy their time on the website or application and not feel as if their time has been wasted by irrelevant items.

2.3 Basics of How Recommendation Systems Work

As previously mentioned, recommender systems use complex algorithms to predict the best item to show any given user. The process often begins when a user

[1] As an example, a tennis player is less likely to be interested in golf clubs than a golfer. The tennis player may prefer to be shown rackets.

first enters or signs up for a website or application. Recommender systems almost always use age, gender, current location, and other similar, basic facts in their algorithms. Many companies will go as far as keeping track of every action a user makes, such as the interaction time of each previous item, the geographical location of each search made, and more. This information is tracked, stored, and sent through a data pipeline to the algorithms being used. Once reaching the recommender system, the data associated with a user will be compared with a filtered set of items and a list will be created with the best items to show the user next. The reason the full list of items is not usually shown is that there can be billions of possible items, and going through each possible item for every user would be too computationally intensive. Thus, some of the previous information is used to narrow down the list of initial items. As an example, a United States resident will not be considered for items that are only available in a foreign nation. The exact way this list is determined depends on the specific algorithm being used, some of which will be covered in depth in Section 3. Once the list is compiled, however, items are simply taken from the list and shown to the user, and, periodically, the list will be updated with new data, restarting the entire process.
Note, however, that the items are not usually taken in the direct order given by the recommender system. This is because often extremely similar items will be ranked closely. Going back to motivator two in Section 2.2, one of the goals of these systems is to show a wider variety of items. Thus, the system will avoid showing items that are too similar, too often. Additionally, if items of near identical similarity are shown in succession, the user may become uninterested and exit the system, going against motivator three.

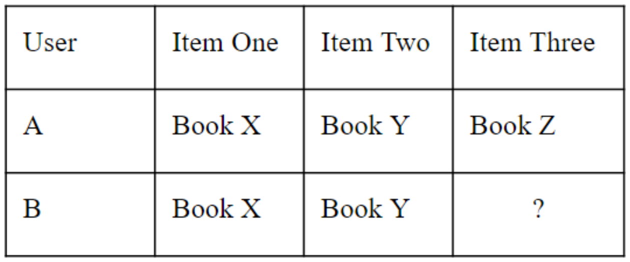
3 CURRENT TECHNOLOGIES

There are a number of common models used in the world of recommendation systems, each of which has its own advantages and disadvantages. In this section, a few of the most common systems will be explained and subsequently discussed.

3.1 Collaborative Filtering Recommendation Systems

One algorithm often used in recommendation systems is the collaborative filtering model. This model uses the history of other users to predict the best items to show a new user. An example of this scenario is shown in Figure 1 where user A has rated three different books highly, and user B has rated two of the same books similarly. Thus, when predicting a book that user B will enjoy, we should predict with relative ease, that user B will like Book Z. This is the essence of collaborative filtering.

A major advantage of this system, as pointed out by Kumar (2022), is that the algorithm requires no metadata of the users or the items. The users can be anyone with



User	Item One	Item Two	Item Three
A	Book X	Book Y	Book Z
B	Book X	Book Y	?

Figure 1: Table Depicting the Usage of Collaborative Filtering Models Between Two Users. (Keaton 2023).

any background, and the items can be any type of rateable item (book, song, movie, game, etc.). The algorithm simply takes users with similar patterns and history and matches items based on this. One potential downfall of this system is when only using collaborative filtering in smaller datasets, this could lead to extremely inaccurate recommendations. Using the same scenario as shown above, imagine that Book X and Book Y are both romance novels, but Book Z is a horror story. There is a chance that user B has no interest in horror books, and thus the system has failed with its recommendation. Over time and with enough data, this problem will be solved, as the system will recognize that if theoretical users C, D, and E did not read or enjoy Book Z, then perhaps it should not be recommended to user B. However, this means that collaborative filtering falls victim to the cold start issue, a widespread problem in recommendation systems that will be discussed in section five.

Another issue that arises in systems that use collaborative filtering is shilling attacks, which are defined by Sundar et al (2020) as "a particular type of attack where a malicious user profile is inserted into an existing collaborative filtering dataset to alter the outcome of the recommender system" (p. 171704). In short, shilling attacks are when a fake user account is created and used to either inflate or deflate the ratings of a particular item. In turn, this rating deflation could cause the item to be recommended to fewer users and have financial ramifications. In naive implementations, the issue can be quickly compounded as described in Schafer et al (2007), as real users may be swayed to change their ratings to match those of the fake users. This can lead to a domino effect, in which otherwise good items spiral to the bottom of the list for recommendations.

Regardless of these issues, collaborative filtering can be an extremely powerful and efficient recommendation algorithm, as companies such as Amazon, Netflix, and Spotify all employ it to some degree, as stated by Kumar (2022).

This process begins with the development of a nearest-neighbors model. A nearest-neighbors model simply groups users (or items) based on several variables. Whenever a new item is added to the model, it is placed within a group based on distances to items around it. A very simple version of this is depicted in a two-dimensional graph in Figure 2.

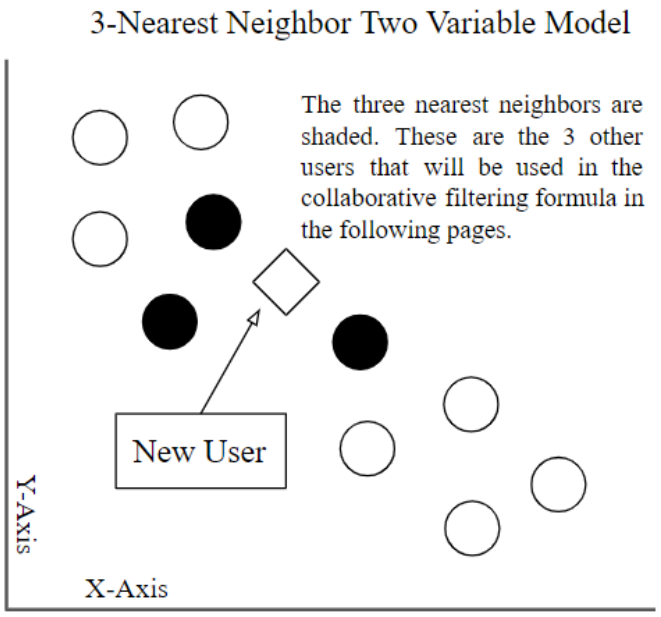

3-Nearest Neighbor Two Variable Model
The three nearest neighbors are shaded. These are the 3 other users that will be used in the collaborative filtering formula in the following pages.
New User
Y-Axis
X-Axis

Figure 2: Graph Depicting the Basis of a K-Nearest Neighbors Model. (Keaton 2023).

However, with large amounts of data, it becomes difficult for the model to efficiently parse through all other users. Thus, a KNN model is used, where K is the number of neighbors accounted for. This model is similar to the nearest-neighbors model, but only a certain number of other neighbors (K = 3 in Figure 2) are taken into account. This allows for a large increase in efficiency, for a potential decline in accuracy. The way this KNN model is created is through the use of a user similarity equation.

Following the KNN model's creation, another prediction formula must be created. There are several equations possible for the basis of collaborative filtering. One of the more simple equations[2] is defined by Schafer et al (2007), shown below in Figure 3, and the definitions of variables used are contained in the index (Figure 4).

This equation attempts to predict the rating of item *i* for a user, *u*. The equation uses the ratings of other similar users, *n* (found by utilizing KNN model explained above), to achieve this goal. The first step in the process is retrieving the current neighbors' similarity score. This score is then multiplied by the current neighbor's rating of the item being predicted, minus the same neighbor's average rating of all items. This process is repeated for each neighbor of user *u* and summed into a total. This total is then divided by the sum of all user's similarities. Finally, the current user average rating is added to the value resulting from the previous division. This process can be hard to follow so the following page will contain a full walkthrough with example data.

The reason that Schafer et al (2007) gives for the averages being included in the equation for both the neighbor and the current user is to account for users who are generally more negative versus generally more positive[3]. As previously mentioned, this formula is a moderate equation regarding complexity. Some equations take many more variables into account. Some of these could include time since ratings occurred, weighted ratings based on an item's overall popularity, or even using other methods of machine learning to find accounts associated with shilling attacks, and subsequently ignoring them in the models. These additional levels of complexity are out of the scope of this paper, and thus will not be discussed in detail.

Section 3.1 has summarized and given one example of a collaborative filtering model, along with discussing some of the potential downfalls of the system, and how these may be solved.

$$pred(u,i) = \bar{r}_u + \frac{\sum_{n \subset neighbors(u)} userSim(u,n) \cdot (r_{ni} - \bar{r}_n)}{\sum_{n \subset neighbors(u)} userSim(u,n)}$$

Figure 3: Equation for Basic Collaborative Filtering Model. (Schafer et al 2007).

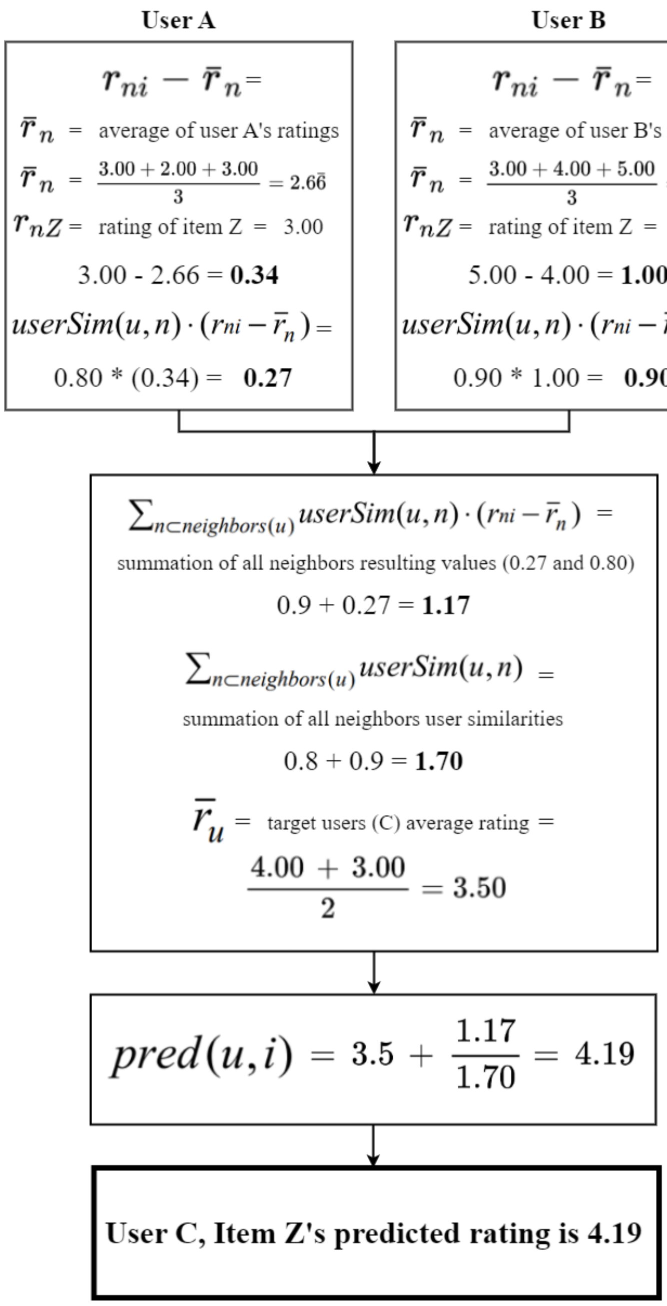
[2] In the article, there are several equations that are worked through. Some become extremely complex and take into account issues that come with collaborative filtering. The equation used in this example is neither the most complex nor the most basic. It is somewhere in the middle, allowing it to demonstrate some of the most important adjustments made to the base equation while still being digestible to most readers.

[3] For some users, a “good” book may be a rating of 3.5/5; whereas, for other users, an equally good book may be a 4.5/5. Even though these two users enjoyed the book the same amount, they gave it different ratings. This equation attempts to take that discrepancy into account.

Below shows a step-by-step walkthrough of three users, their ratings of previous items, and a prediction of a future item for one of the three users. The example data is given in Figure 5.

User	Item X Rating	Item Y Rating	Item Z Rating
A	3	2	3
B	3	4	5
C	4	3	?

Figure 5: Table of Example Data for Collaborative Filtering Model . (Keaton 2023).



User A

$r_{ni} - \bar{r}_n =$

$\bar{r}_n$ = average of user A's ratings

$\bar{r}_n = \frac{3.00 + 2.00 + 3.00}{3} = 2.6\bar{6}$

r_{nZ} = rating of item Z = 3.00

3.00 - 2.66 = **0.34**

$userSim(u,n) \cdot (r_{ni} - \bar{r}_n) =$

0.80 * (0.34) = **0.27**

User B

$r_{ni} - \bar{r}_n =$

$\bar{r}_n$ = average of user B's ratings

$\bar{r}_n = \frac{3.00 + 4.00 + 5.00}{3} = 4.00$

r_{nZ} = rating of item Z = 5.00

5.00 - 4.00 = **1.00**

$userSim(u,n) \cdot (r_{ni} - \bar{r}_n) =$

0.90 * 1.00 = **0.90**

$\sum_{n \subset neighbors(u)} userSim(u,n) \cdot (r_{ni} - \bar{r}_n) =$

summation of all neighbors resulting values (0.27 and 0.80)

0.9 + 0.27 = **1.17**

$\sum_{n \subset neighbors(u)} userSim(u,n) =$

summation of all neighbors user similarities

0.8 + 0.9 = **1.70**

$\bar{r}_u$ = target users (C) average rating =

$\frac{4.00 + 3.00}{2} = 3.50$

$$pred(u,i) = 3.5 + \frac{1.17}{1.70} = 4.19$$

User C, Item Z's predicted rating is 4.19

3.2 Content-Based Recommendation Systems

Another common and naive model used in recommendation systems is the content-based algorithm. This recommender takes the opposite approach of collaborative filtering models. While collaborative filtering models find similar users and recommend items based on those similarities, content-based recommenders find the items a user likes and recommends more items that are similar to the original. This is briefly shown in Figure 6 to the right.

There are countless pros and cons to this model, however, perhaps the most blatant negative is how susceptible the model is to the cold start problem– a challenge that will be further discussed in Section 5.

Additionally, as explained throughout chapter four of Ricci (2015), this method is only at its peak with text-based data. Content-based models that work with items such as images, videos, or other forms of media require those other media types to first be processed and categorized as text[4].

This is because most content-based filtering uses a method of filtering called Term Frequency-Inverse Document Frequency (TF-IDF). The goal of this method is to collect the frequent terms of one item while deciphering how common the term is in general.

As a simple example, imagine you are reading an article about Ancient Egypt– some possible unique frequent terms in this article would be "Pyramid", "Monarchy", or "Cleopatra". This issue arises when the actual most common words are terms such as "the", and "or". We don't want other articles that have a lot of the word "the", that most readings! We want only readings that have words specific (or mostly specific) to those of Ancient Egypt. This is where the "IDF" portion comes in. This section calculates the overall frequency of a term compared to *all* documents and compares that with the target item. This means that only words that are usually not frequent ("Cleopatra"), will receive a high score, and commonly used words across all articles ("the"), will receive a low score.

The equation for TF-IDF is relatively simple and is given below in Figure 7.

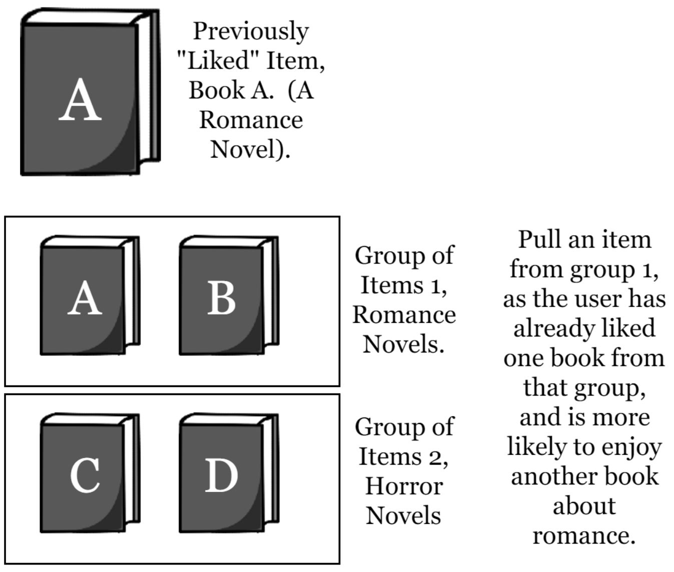

A
Previously "Liked" Item, Book A. (A Romance Novel).
A
B
Group of Items 1, Romance Novels.
C
D
Group of Items 2, Horror Novels
Pull an item from group 1, as the user has already liked one book from that group, and is more likely to enjoy another book about romance.

Figure 6: Visualization Giving Basic Representation of Content-Based Model. (Keaton, 2023)

$$TF_t \times \log \frac{N}{DF_t}$$

TF_t = Term Frequency (how many times the word 't' appears in the given document)

N = Number of Documents

DF_t = Document Frequency (how many documents does the word 't' appear in)

Figure 7: Formula for the TF-IDF Used in Content-Based Models. (Keaton, 2023)

[4] As an example, a picture of a red train in the desert is not enough. Either an individual needs to label this image as a "red train in the desert", or another form of machine learning needs to do so automatically– but this is outside the scope of the paper.

Due to the usage of logarithms, words that are extremely common in the English language will not be given a score. For example, if there were 100 documents, and all 100 documents had a word appear, this would lead to a score of 0, as the logarithm of 1 (100/100), is 0. This means that no matter how many times this term was to appear in any given document, its score would remain 0.

Alternatively, given an uncommon word, such as the above example of "Pyramid", and given 100 documents, it may only appear in 10. This leads to a score of 10 times the number of times the word appears in the target document.

This simple formula is an extremely effective basis for finding similar text-based items, and thus recommending text-based data extremely well. The next step of the process is to put all items into a matrix of their words, removing any words that are too common (above a certain threshold of the TF-IDF value), or too rare (very few other documents have them, and thus not useful for recommendations), and removing them.

Finally, using a nearest-neighbors model (similar to what is described in section 3.1), create groups of items and categorize them.

An example of this process is given in the 2016 paper written by Titipat Achakulvisut et al. The paper describes the above process and adds the additional step of using a Latent Semantic Analysis (LSA) to preprocess the data more and create a 2-dimensional set of data that can be used for grouping in the nearest neighbors model.

Achakulvisut et al. used his proposed model to create an example for the conference he was attending, in which he had different researchers pre-group articles written into several groups such as "Neuroscience". These groups were then given letter symbols and the articles were plotted on a two-dimensional graph,

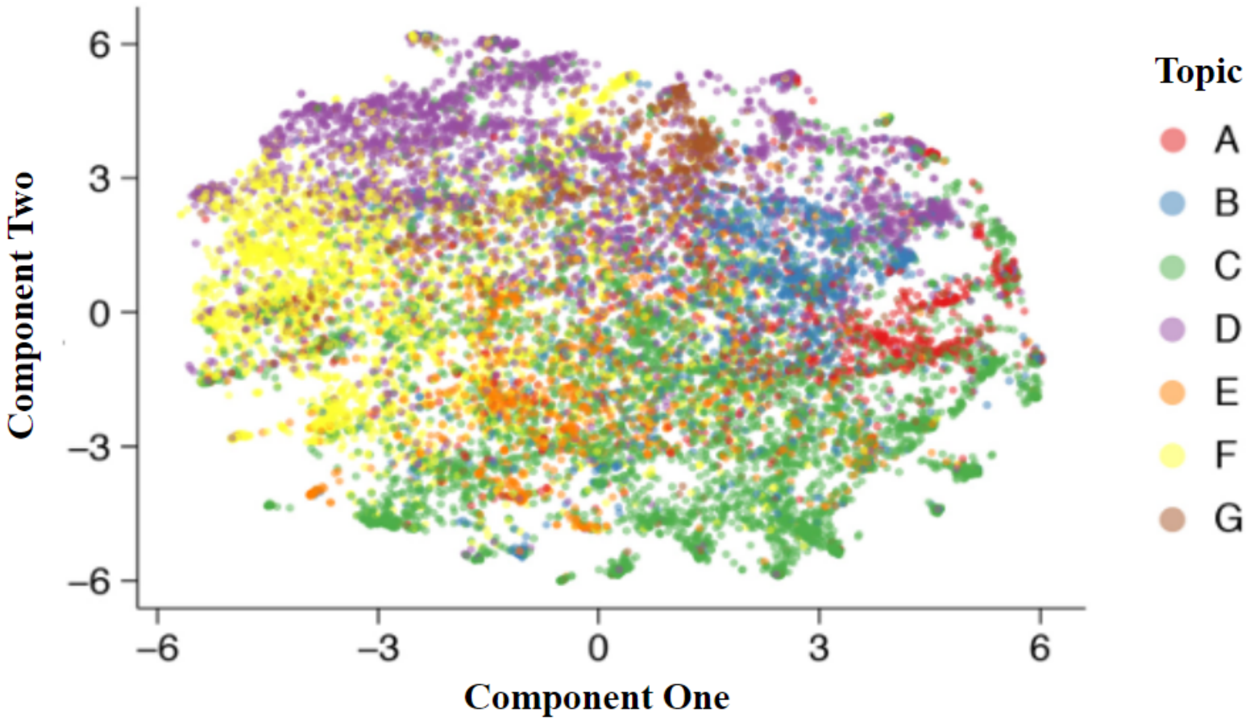


Figure 8: Vector Representation of Documents (Achakulvisut et al., 2016)

representative of the nearest-neighbors model created in the process. This was done to visualize how manually grouped data (the letters/colors) compared to the groupings created by the algorithm. The result is shown above in Figure 8.

As can be seen, the example model does a good job of clustering the topics using TF-IDF, and implementing a content-based filtering model from this point is trivial. The next step is to find the items that are the closest distance to past "liked" items and recommend those to the user.

This model has its own set of issues though. In the same way as collaborative filtering models, content-based models also struggle with the cold-start problem, and without a history of likes and dislikes from a user, the model will not know what to recommend. More uniquely, though, content-based models struggle with the issue of never recommending new items to a user. Once a user gets a base set of "likes", the system will always try to recommend the same items. Expanding on the example from Figure 6 on page 7, if a user starts rating romance novels highly, they will *never* be recommended horror novels, or other genres in general. More generally this problem could be called overfitting– the idea that only recommending the closest possible items could lead to an overall worse algorithm. Another clear issue is that these systems are natively language-locked. Translations can be implemented, but basic versions cannot be implemented worldwide, which is a non-issue for collaborative filtering.

3.3 Hybrid Recommendation Systems

One method of attempting to deal with the negatives of each of the above-described recommendation models, is to use a hybrid of models. There are many ways to go about this, many of which were explained by Jeffrey Chiang in his 2021 article 7 Types of Hybrid Recommendation Systems, some of which are described below. The idea of a hybrid model is to take two or more distinct models (commonly content-based and collaborative filtering) and combine them in some fashion to maximize the positives of each while negating the cons as much as possible. Below are three of the most common forms of hybrid models described.

1) The simplest. and perhaps the most common form of a hybrid recommendation system is the weighted recommendation system. In this method, two other systems have distinct weights that add up to a sum, and the final model takes portions of each to predict scores for a user. For example, 50% comes from a collaborative model, and 50% comes from a content-based model. Alternatively, depending on the data, the weights could lean 70/30 in some direction. This model, and many other hybrid methods may have performance.

2) Another form of a hybrid model is the switching model, in which the recommender systems recognize a situation and apply unique models on a case-by-case basis. This allows recommendation systems to be used during their strengths, and avoided during their weaknesses.

3) The third most common hybrid model is the mixed model. This model allows all (two or more) models to work independently, and compare results at the end, creating one

combined list of predicted items that is used as a master list. This method is graphically similar to the weighted model, however, the end result will be different. This model can also struggle with performance, as many models must be executed for every user for each new recommendation list created.

3.4 Other Common Recommendation Systems

Countless other models are used for recommendation systems, many of which are extremely sophisticated and are not relevant to the core goals of this paper. These include complex clustering models, bayesian networks, neural networks, or even simpler concepts such as a decision tree. Each of these alternative models has positives and negatives that make them suitable for one situation or another, which is largely why hybrid models were developed.

Within each category is an incredible amount of research and testing to minimize problems any given model may have. Due to this research, countless new technologies have been developed and improved upon, leading to the incredibly efficient models seen today in companies such as TikTok, Netflix, and Google Advertisements.

4 EMERGING TECHNOLOGIES

As stated, there is an immense amount of research occurring around the globe to create the best possible recommendation algorithm. Some research aims to solve current problems, such as the aforementioned "cold start" issue, and others are attempting to create entirely new standalone systems, such as TikTok's Monolith. This section will discuss some of the most exciting new developments, and how they may affect the future of social media users.

4.1 Cold Start Issue

Perhaps the most consistent issue among all recommendation systems is the cold start problem. As previously mentioned, the cold start problem is the idea that whenever a new item or user is added to a model, there is no historical data for that item or user, and thus no strong method for recommending them. As an example, in the aforementioned model for collaborative filtering in Figure 3, if there is no way to calculate the user similarity or a user's past average ratings, the model is essentially guessing. This will remain the case for some time (depending on the platform). For applications such as TikTok, where users may go through 5+ items a minute, the model will get sufficient data very quickly. However, for a platform such as Netflix, where a user may only watch and finish one show a week, it can become much more difficult.

Research conducted by Shilong Liu et al. found a potential solution to this cold start issue. Their recommendation is to leverage a combination of meta-learning and attention-based learning to combat items and users with little to no prior data. Meta-learning is a concept in machine learning that is essentially a model "learning to learn". The model will use the output of its own experiments to change itself to better suit the needs of the problem. Attention-based learning is similarly a model learning what sections of data to pay attention to during calculations. This allows for potentially irrelevant data to be entirely ignored.

Liu et al. initially discuss other methods of solving the cold start problem. First, the paper mentions using the “perspective of bandits… to embed the preference of new users in a social network. (pg. 2). Multi-armed bandits are in short, when there exist multiple choices for some question, where there is an optimal choice to be made. One example given in an article by Optimizely is a gambler in front of a line of slot machines– one machine will have the biggest payout, however, this machine is not yet known to the gambler. There are varied solutions to this problem, again described in the Optimizely article, such as the Epsilon-Greedy algorithm, in which the option with the highest historical payout is provided, except for on some occasions when a random option is chosen. Another is the upper confidence bound solution, in which all options are sorted by their highest possible payoff given historical data. Finally, there is the Thompson Sampling solution, which simply says to choose the highest expected value model in any given distribution.

Another method Liu et al. discuss is the matrix factorization method. Liu et al. briefly describe this message stating that "It [matrix factorization] is to find out user item latent factors based on the available users’ preferences.” (pg. 3).

Liu et al. believed that these methods were not strong enough solutions to the cold start problem, however, and aimed to develop a new method. As previously stated, this new method combines meta-learning and attentive recommendation models in order to provide items to new users, and vice versa.

Their proposed method begins with the use of a model-agnostic meta-learning (MAML) model. Model-agnostic simply means that the input that is created can be used as an input for different types of models. The alternative is a model-specific input, in which the data can only be used to train and test one type of machine learning model. This is important for the second portion of MAML, which is meta-learning. Meta-learning, as previously mentioned, is the idea of a model “learning to learn”. In the context of the proposition by Liu et al., meta-learning is used to decide which type of model to use for any given batch of users. What this means is that every user or item may be on a different type of recommendation system, depending on the amount and type of data that the system has access to.

As an example, a user who rates every item they receive may be more suited for a content-based model, whereas a user who has never rated an item may be more conducive to a collaborative filtering algorithm.

The MAML works by testing a set of different algorithms ($\mathcal{F}$) on a parameter (ϕ) with a support set of data (D^S) for each user (u_i) in a set, and then choosing the algorithm with the lowest test loss gradient for a query/test set of data (D^Q) ($\mathcal{L}$). This algorithm is shown below in Figure 9.

$$\min_{\varphi} \sum_{u_i} \mathcal{L}_{D^Q_{u_i}}(\theta_{u_i}) = \sum_{u_i} \mathcal{L}_{D^Q_{u_i}}(\mathcal{F}(D^S_{u_i}, \varphi))$$

Figure 9: Base Formula for MAML Algorithm (Liu et al., 2023)

In short, the MAML model tests many possible algorithms with each user and chooses the initial model that works the best for each user.

This process ends with an output of an initial model that can then be tweaked to be better than the initial option. The next step of the process is using the attention network in order to give the correct level of weighting to the different factors used. The architecture for the process is shown in Figure 10.

The inputs to the network ($E_u{}^u$ and $E_u{}^v$) are vectors for the users and items that are then put through the algorithm to correctly weigh each of the user's and items' contents.

This Attentional Meta-Learned User preference estimator (AMeLU) proposed by Liu et al. proves to be extremely efficient in comparison with some other common solutions and is shown in the results section of the paper. Figures 11 and 12 show both the MAE and RMSE values compared to the algorithm's counterparts. Mean absolute error values (MAE) are better when lower, as it measures the average error margin of a model when making predictions. As shown, the AMeLU and MeLU models perform significantly better than the PPR and Wide&Deep models at all cold rates, especially at cold rates approaching 1.0. Root-mean-square error behaves similarly to MAE and thus has a similar graph.

Some of the potential issues with the model though are its complexity along with time complexity, as the time complexity of the attention network alone is O(n2·d) as stated on page 7 of the paper.

Overall, the AMeLU model performs better than the other three common models, even if only marginally so over the MeLU model. Even though there are still countless improvements to be made in the cold start problem, Liu et al. propose a very strong model to solve the issue, albeit with some concerns in terms of efficiency and complexity.

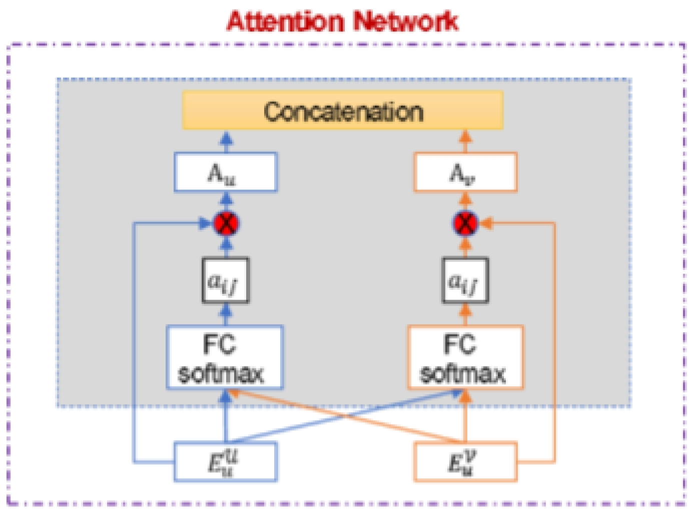


Figure 10: Architecture of Attention Network (Liu et al., 2023)

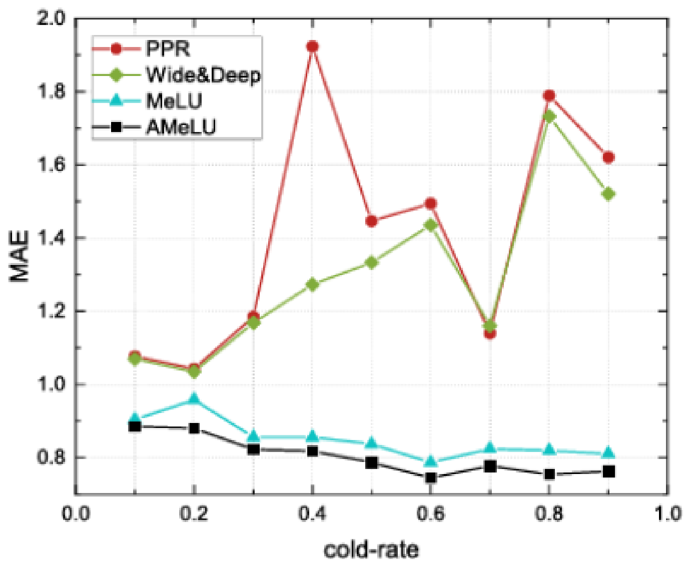


Figure 11. The MAE of four models over different cold-start levels. (Liu et al., 2023)

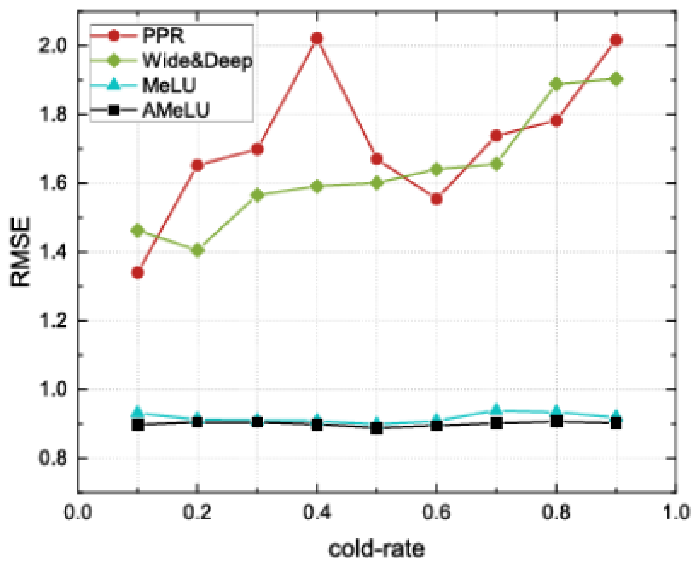


Figure 12. The RMSE of four models over different cold-start levels. (Liu et al., 2023)

4.1 Concept Drift

Another pressing issue faced in the modern world of recommendation systems is concept drift. In short, concept drift is the idea that a user's interests will shift over time– the most obvious case being that of political positions. This is more technically defined in Gama et al’s. article, A survey on concept drift adaptation, where they state “Concept drift primarily refers to an online supervised learning scenario when the relation between the input data and the target variable changes over time.” (2014, pg. 1) A naive solution to this would be to remove data after a certain amount of time has been reached, whether it be a week or a year, there would be a hard cut-off that removes data from the model for a user.

This idea can of course be improved upon. Some of these methods are discussed in Gama et al’s. 2014 article. They offer up two main aspects of concept drift improvement, being memory and ‘forgetting’. For memory, there is single and multiple memory, with single memory being storing each data point individually. Multiple on the other hand, will store many single data points within a model, and these models can then be used and updated. The former can struggle to update quickly enough with concept drift, while the latter improves upon this.

Within the memory groups, there is a sub-question of whether data should exist in a fixed sized window or variable size window. For example, should exactly 100 data points always be used, or should any data points in the past two weeks be used. Or some combination (up to 100 data points, but must be within the last two weeks).

Following this is the question of how, and when, to forget data. There are a number of options, but the two broad groups are to abruptly forget data, meaning to take it completely out of the models without leaving any reminisce, or to gradually forget data, meaning have the data matter less and less in the model over time.

All of the above options can work, and do work for different situations, and so there is no true one size fits all approach. However, as we will see in the upcoming section, some large social media developers tend to use the concept of gradually forgetting, but with heavy bias on recency.

4.2 Monolith

Although the cold start issue is a large issue for all recommendation systems, a select few have more pressing problems. These issues are saved for the giants of the tech and social media industry, such as TikTok, Google, Instagram, Netflix, etc. These problems relate to the storage and speed of their massive models that encounter billions of data points a day. Preliminary solutions to these problems are proposed by Liu et al., in their 2023 paper titled Monolith: Real-Time Recommendation System With Collisionless Embedding Tables. The paper is an analysis of the Monolith recommendation system, used by Bytedance, the company that owns TikTok. Within the introduction of the paper, the authors bring about two primary points they have found within modern recommendation systems in which changes must be made. They are as follows:

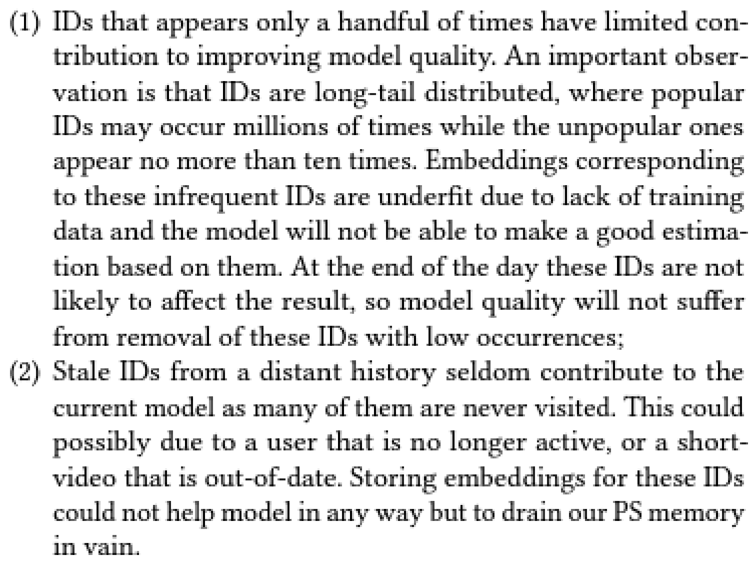
“(1) The features are mostly sparse, categorical, and dynamically changing.

(2) The underlying distribution of training data is non-stationary, aka Concept Drift [discussed section 4.1].” (Liu et al, pg. 1)

These two points lead to two obvious improvements that can be made to the model, being an increase in the number of times per session a user's model is updated, and the speed in which data can be processed by the model for these more consistent changes.

The first step Liu et al. take is to create a collisionless hash table. Hashmaps using key-value pairs are the main method TikTok uses for storage and thus, they need a fast and memory-efficient means to store and access data. Hashmaps, in layman's terms, are a method of storing data that allows for faster look-ups of regularly accessed data than standard data types in programming. Their chosen solution is the cuckoo hashmap (CITE). This hashing method manages to have a collisionless insertion, which is important to avoid any overwriting of previous data. Additionally, the algorithm maintains the pros of a standard hashtable, such as an O(1) time complexity (meaning the worst case is constant time) lookup and deletion, along with amortized (usually) O(1) insertion.

Liu et al. continue their improvements, though. The next issues that arise are:


(1) IDs that appears only a handful of times have limited contribution to improving model quality. An important observation is that IDs are long-tail distributed, where popular IDs may occur millions of times while the unpopular ones appear no more than ten times. Embeddings corresponding to these infrequent IDs are underfit due to lack of training data and the model will not be able to make a good estimation based on them. At the end of the day these IDs are not likely to affect the result, so model quality will not suffer from removal of these IDs with low occurrences;

(2) Stale IDs from a distant history seldom contribute to the current model as many of them are never visited. This could possibly due to a user that is no longer active, or a short-video that is out-of-date. Storing embeddings for these IDs could not help model in any way but to drain our PS memory in vain.

In short, what Liu et al. are saying is that first, some items will rarely be seen, and thus having them take up the same amount of memory as items with millions of views and interactions is pointless (they won't contribute to the model significantly) and secondly, that old items (in some cases years), also are less relevant for recommending new content to users. As an example, a video from a 2019 trend is less likely to be of interest to a user than a video about a trend from right now.

The solutions implemented for these two problems are as expected. First, new items are filtered by the probability of them being useful to the models, and second, all items are set to be inactive after a certain duration.

To summarize the first issue and solution proposed by Liu et al., there is a need for a fast and memory-efficient means to handle data for mass recommendation systems such as the one used in TikTok, and Liu et al. have discovered a way to do this, while also cleverly avoiding any data that is irrelevant for any reason.

The next goal Liu et al. bring up is the goal of attempting to have consistent and live changes in user models. They solve this issue seamlessly with a number of steps and procedures, beginning with the separation of batch training and online training. They describe this separation, with batch training being offline and similar to standard TensorFlow training. Liu et al. state that:

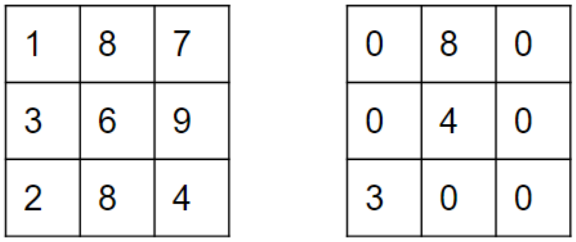
"Batch training is useful for training historical data when we modify our model architecture and retrain the model." (Liu et al. pg. 4)

What this means is that the batch training section is primarily useful for large changes in a users model– when changes more than just slight tweaks to parameters need to be made, and the changes will take longer to compute.

Alternatively, Liu et al. also describe an Online training stage, which consists of live changes to a user model, that allows for small changes to be made while a user is on the app. Liu et al. describe this process more in-depth as:

"a training worker consumes realtime data on-the-fly and updates the training PS [Parameter Server]. The training PS periodically synchronizes its parameters to the serving PS, which will take effect on the user side immediately. This enables our model to interactively adapt itself according to a user's feedback in real time." (Liu et al. pg. 4)

In simpler terms, this is saying that a training recommendation model will make changes as a user is using the application, and periodically these changes will be sent to the live recommendation model that the user is currently being recommended from. The next step in updating models live is to do so in an efficient manner. As stated on page 5 of Liu et al's. paper, user models tend to be multiple terabytes in size. Updating a model of this size, for each user, every minute, is simply unachievable. Liu et al. solve this problem by separating variables into sparse and dense variables. Sparse embedding tables are simply those with little data, being dominated by 0's, whereas dense variables are those that are mostly non-zero values. This concept is shown below in Figure 13.



1	8	7
3	6	9
2	8	4

0	8	0
0	4	0
3	0	0

Figure 13: Tables Depicting the Difference Between Sparse and Dense Embedding Tables. (Keaton 2023).

As Liu et al. state, sparse parameters dominate the size of models, even though they will be updated much less often. As a naive example, imagine embedding tables that have the goal of calculating what a given article is about. One table about professional sports may have counts of words such as "ball", "score", or "referee". However, another table may keep count of the topic of World War II, in which words such as "war", "tank", or "politics" may be used. If the article is in fact, about professional sports, only that table will be heavily filled, and most other tables on other topics will have little to no data. Due to the fact that there are countless more topics the article doesn't apply to, many more tables will be left nearly or entirely empty than those that will be filled. This is similar to the concept of why recommendation systems will have more sparse variables than dense ones. Liu et al. leverage this fact to only occasionally update a user's dense variables, and tend to update a few sparse variables with each live model update, instead. Liu et al. claim this outright when they state:

"We push the subset of sparse parameters whose keys are in the touched-keys set with a minute-level interval from the training PS to the online serving PS." (pg. 5)

Additionally, it is discussed that dense parameters are only updated daily, allowing for updates to the larger portions of user models on a consistent basis, while not hindering the network to an extreme degree. Liu et al. further optimize this process with the following:

"... dense parameters, since they are synchronized daily, we choose to schedule the synchronization when the traffic is lowest (e.g. midnight)." (pg. 8)

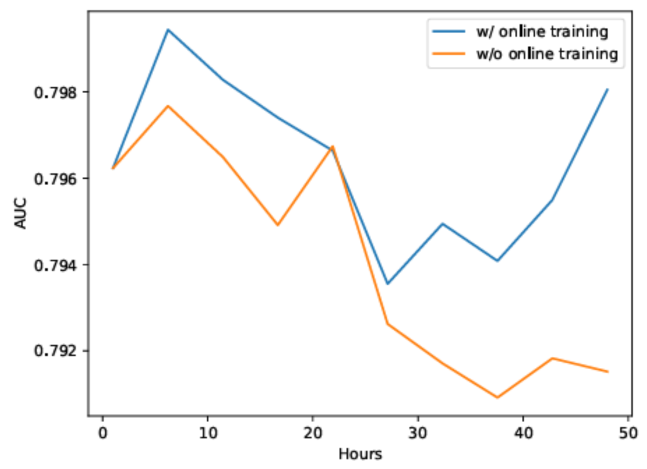
meaning that dense parameter updates for all users of a region are only implemented when network traffic is at a daily low for that area, and thus making the overall process more efficient and server-friendly.

The two aforementioned changes to standard TensorFlow neural networks allow for massive optimizations and an incredibly powerful recommendation system. This is briefly proven in the evaluation section of the paper. In Figure 14 below, some analysis is provided for different live model update times (5hr, 1hr, 30min), and how these intervals affect a model's performance over the course of 50 hours. The results are clear; a shorter live model update time means a slight increase in model performance. As stated in a previous quote, the production model of Monolith uses a one-minute interval, which means the model is performing at even better rates than those shown in the figures.

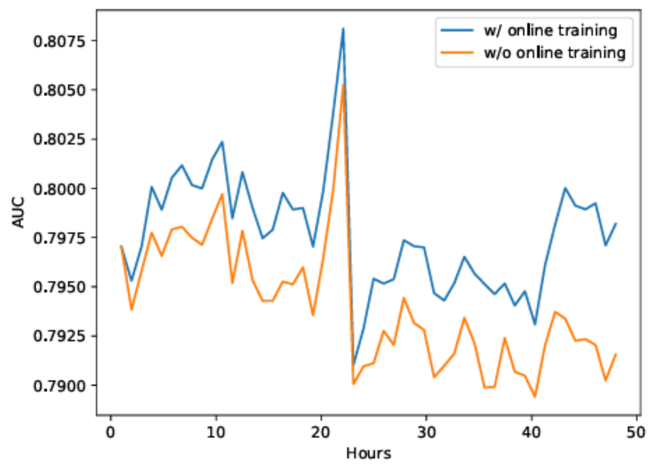
5 CHALLENGES, FUTURE TECHNOLOGIES, AND CONCLUSION

Recommendation systems are currently faced largely with the same major issues that face all technological research–speed and size. Many optimizations outside of those mentioned in this paper are centered heavily on making the algorithms faster, and work as well while taking up less space. These topics will likely never truly be gone, as there will always be room for improvement in both aspects. Outside of this, comes the challenge of more efficiency and accuracy, some of which was covered above. With the increase of the general effectiveness of recommendation systems, though, comes perhaps the most daunting challenge of all, being the social perception and ethical use of recommendation systems.

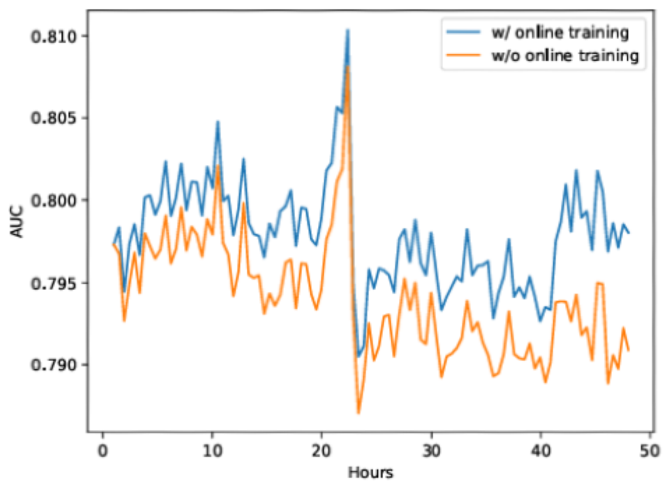
As is covered in my paper An Analysis of Recommendation System's Effect on Society and the Engineers Involvement (2023), recommendation



(a) Online training with 5 hrs sync interval



(b) Online training with 1 hr sync interval



(c) Online training with 30 min sync interval

Figure 14: Three Tables Showing Difference in Performances Based on Intervals (Liu et al. 2022)

algorithms are potentially a major reason for the decline of mental health worldwide. With social media addiction being in the spotlight, developers need to come up with a way to keep users interacting with their platforms, while decreasing the negative consequences that come with extremely powerful systems. Recommendation systems will continue to improve at exponential rates, however it is the job of the researchers and engineers to make sure they create software that is useful for users, rather than detrimental to their health and wellbeing.

INDEX

Symbol	Definition
u	Current user (the diamond in Figure 2).
i	Current Item being predicted
n	The current neighbor being calculated (one of the circles in Figure 2).
userSim(u, n)	The user similarity equation. Outputs range from -1.0 (complete disagreement) to 1.0 (complete agreement).
$\bar{r}_u$	The average rating of all items by the current user.
$r_{ni} - \bar{r}_n$	The current items rating by the current neighbor, minus the average rating of all items by the current neighbor
$\sum_{n \subset neighbors(u)} FUNC$	The sum of a function FUNC applied to each *n*, where each *n* is a neighbor of *u*.

Figure 4: Variable Definitions for Equation Used in Figure 3. (Keaton 2023).

REFERENCES

Achakulvisut T, Acuna DE, Ruangrong T, Kording K. (2016). Science Concierge: A Fast Content-Based Recommendation System for Scientific Publications. PLOS ONE 11(7): e0158423. https://doi.org/10.1371/journal.pone.0158423

Achakulvisut T, Acuna DE, Ruangrong T, Kording K. (2016). Figure 8: Vector Representation of Documents. [Figure 8]. Science Concierge: A Fast Content-Based Recommendation System for Scientific Publications. PLOS ONE 11(7): e0158423. https://doi.org/10.1371/journal.pone.0158423

Cabrera-Sánchez, J.-P., Ramos-de-Luna, I., Carvajal-Trujillo, E., & Villarejo-Ramos, Á. F. (2020). Online recommendation systems: factors influencing use in e-commerce. *Social Media Influence on Consumer Behaviour*, 12(21), 1-15. http://dx.doi.org/10.3390/su12218888

Chiang, Jeffrey. (June, 2021). 7 Types of Hybrid Recommendation System. Medium. Retrieved November, 2023 from https://medium.com/analytics-vidhya/7-types-of-hybrid-recommendation-system-3e4f78266ad8

Gama João, Žliobaitė Indrė, Bifet Albert, Pechenizkiy Mykola, and Bouchachia Abdelhamid. 2014. A survey on concept drift adaptation. ACM Comput. Surv. 46, 4, Article 44 (April 2014). https://doi.org/10.1145/2523813

Jiawei Chen, Hande Dong, Xiang Wang, Fuli Feng, Meng Wang, and Xiangnan He. 2023. Bias and Debias in Recommender System: A Survey and Future Directions. ACM Trans. Inf. Syst. 41, 3, Article 67 (July 2023), 39 pages. https://doi.org/10.1145/3564284

Keaton, B. (2023). Table Depicting the Usage of Collaborative Filtering Models Between Two Users. [Figure 1]. A State of The Art in Machine Learning Recommendation Systems (Unpublished undergraduate thesis). School of Engineering and Applied Science, University of Virginia. Charlottesville, VA.

Keaton, B. (2023). Depicting the Usage of Collaborative Filtering Models Between Two Users. [Figure 2]. A State of The Art in Machine Learning Recommendation Systems (Unpublished undergraduate thesis). School of Engineering and Applied Science, University of Virginia. Charlottesville, VA.

Keaton, B. (2023). Table of Example Data for Collaborative Filtering Model. [Figure 5]. A State of The Art in Machine Learning Recommendation Systems (Unpublished undergraduate thesis). School of Engineering and Applied Science, University of Virginia. Charlottesville, VA.

Keaton, B. (2023). Visualization Giving Basic Representation of Content-Based Model. [Figure 6]. A State of The Art in Machine Learning Recommendation Systems (Unpublished undergraduate thesis). School of Engineering and Applied Science, University of Virginia. Charlottesville, VA.

Keaton, B. (2023). Formula for the TF-IDF Used in Content-Based Models. [Figure 7]. A State of The Art in Machine Learning Recommendation Systems (Unpublished undergraduate thesis). School of Engineering and Applied Science, University of Virginia. Charlottesville, VA.

Ko, H., Lee, S., Park, Y., & Choi, A. (2022). A survey of recommendation systems: recommendation models, techniques, and application fields. *Electronic Solutions for Artificial Intelligence Healthcare Volume II*, 11(1), 1-48. http://dx.doi.org/10.3390/electronics11010141

Kumar, Ajitesh. (May, 2023). Recommender Systems in Machine Learning: Examples. Retrieved June, 2023 from https://vitalflux.com/recommender-systems-in-machine-learning-examples/#:~:text=Collaborative%20filtering%20recommender%20system,-Another%20type%20of&text=Collaborative%20filtering%20is%20one%20of,Amazon%2C%20Netflix%2C%20and%20Spotify

Liu, S., Liu, Y., Zhang, X., Xu, C., He, J., & Qi, Y. (2023). Improving the Performance of Cold-Start Recommendation by Fusion of Attention Network and Meta-Learning. Electronics, 12(2), 376. MDPI AG. Retrieved from http://dx.doi.org/10.3390/electronics12020376

Liu, S., Liu, Y., Zhang, X., Xu, C., He, J., & Qi, Y. (2023). Base Formula for MAML Algorithm. [Figure 9]. Improving the Performance of Cold-Start Recommendation by Fusion of Attention Network and Meta-Learning. Electronics, 12(2), 376. MDPI AG. Retrieved from http://dx.doi.org/10.3390/electronics12020376

Liu, S., Liu, Y., Zhang, X., Xu, C., He, J., & Qi, Y. (2023). Base Formula for MAML Algorithm. [Figure 10]. Improving the Performance of Cold-Start Recommendation by Fusion of Attention Network and Meta-Learning. Electronics, 12(2), 376. MDPI AG. Retrieved from http://dx.doi.org/10.3390/electronics12020376

Liu, S., Liu, Y., Zhang, X., Xu, C., He, J., & Qi, Y. (2023). The MAE of four models over different cold-start levels. [Figure 11]. Improving the Performance of Cold-Start Recommendation by Fusion of Attention Network and Meta-Learning. Electronics, 12(2), 376. MDPI AG. Retrieved from http://dx.doi.org/10.3390/electronics12020376

Liu, S., Liu, Y., Zhang, X., Xu, C., He, J., & Qi, Y. (2023). The RMSE of four models over different cold-start levels [Figure 12]. Improving the Performance of Cold-Start Recommendation by Fusion of Attention Network and Meta-Learning. Electronics, 12(2), 376. MDPI AG. Retrieved from http://dx.doi.org/10.3390/electronics12020376

Liu, Zhuoran, et al .(September, 2022). Monolith: Real Time Recommendation System With Collisionless Embedding Table. *arXiv*.

Optimizeley. Multi-Armed Bandit. Retrieved December, 2023 from https://www.optimizely.com/optimization-glossary/multi-armed-bandit/

Ricci, F., Rokach, L., & Shapira, B. (2015). Recommender systems: introduction and challenges. *Recommender Systems Handbook*. 1-32. https://doi.org/10.1007/978-1-4899-7637-6_1

Schafer, J.B., Frankowski, D., Herlocker, J., & Sen, S. (2007). Collaborative Filtering Recommender Systems. *The Adaptive Web, Lecture Notes in Computer Science,* (4321). https://doi.org/10.1007/978-3-540-72079-9_9

Schafer, J.B., Frankowski, D., Herlocker, J., & Sen, S. (2007). Equation for Basic Collaborative Filtering Model. [Figure 3]. *The Adaptive Web, Lecture Notes in Computer Science,* (4321). https://doi.org/10.1007/978-3-540-72079-9_9

Sundar A.P, Li F., Zou X., Gao T., Russomanno E.D., "Understanding Shilling Attacks and Their Detection Traits: A Comprehensive Survey," in IEEE Access, vol. 8, pp. 171703-171715, 2020, doi: 10.1109/ACCESS.2020.3022962.

X. Dai, Y. Cui, Z. Chen & Y. Yang. (2018). A Network-Based Recommendation Algorithm*. 3rd International Conference on Computational Intelligence and Applications (ICCIA)*, 2018, 52-58. http://dx.doi.org/10.1109/ICCIA.2018.00018

Yifan Wang, Weizhi Ma, Min Zhang, Yiqun Liu, and Shaoping Ma. (2023). A Survey on the Fairness of Recommender Systems. *ACM Trans*, (41). https://doi.org/10.1145/3547333