Weighing Dynamic Availability and Consumption for Twitch Recommendations

Edgar Chen
Twitch
San Francisco, CA, USA
chenedga@twitch.tv

Mark Ally
Twitch
San Francisco, CA, USA
mally@twitch.tv

Eder Santana
Twitch
San Francisco, CA, USA
edesanta@twitch.tv

Saad Ali
Twitch
San Francisco, CA, USA
sali@twitch.tv

ABSTRACT

Twitch is a community driven live streaming video service which uses recommendations trained on implicit feedback data. While this data is available in large quantities, it is subject to various biases and shortcomings. In particular, Twitch watch history data is heavily affected by the unpredictable and irregular behavior of users and streams going online and offline. Two resulting issues are: accounting for negative examples caused by users and channels not being available simultaneously, and understanding the relative scale of importance of positive watch time feedback. In this paper, we propose two methods of loss weighting: one method to address the availability of channels, and another method to adjust for the preferences implied by differing amounts of minutes watched. We discuss the methods, offline experimentation, and the results of adjusting evaluation metrics to fit the new loss weighting methods. Finally, we demonstrate success in a sitewide A/B test that increased recommended minutes watched by 7.9%.

CCS CONCEPTS

• **Information systems** → **Recommender systems**.

KEYWORDS

Recommender Systems

ACM Reference Format:
Edgar Chen, Mark Ally, Eder Santana, and Saad Ali. 2022. Weighing Dynamic Availability and Consumption for Twitch Recommendations. In *Proceedings of The 2nd International Workshop on Online and Adaptive Recommender Systems (KDD-OARS '22)*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Twitch is a community driven video service that hosts live, interactive broadcasts, and their communities. Every day millions of creators broadcast their content to tens of millions of visitors, resulting in billions of minutes watched. A vast majority of these viewing minutes are for live content. Viewers are often interested in a particular live stream because they like some combination of the streamer, the content (e.g. the particular game that is being played), and interaction with the community. On Twitch, viewers have many ways to get to the content they want to watch that are personalized and tailored by Twitch's recommendation system.

Twitch's Recommendation Problem: Twitch's focus is on live video streams, which means that users have a selection of live streams hosted on various channels. Each channel is operated by a streamer, and streams often consist of a live video screen capture of the streamer's computer doing some activity (e.g. playing a video game, drawing digital art), a camera overlay which shows the streamer, and audio of the streamer speaking to the audience. The audience is also able to communicate with each other and with the streamer in an interactive chat window (see Figure 1 for an illustration). Due to the emphasis on live content, there are certain characteristics that distinguishes Twitch's recommendation problem from a traditional recommendation setting (e.g. recommending VOD content) commonly addressed in academia and industry (e.g. Netflix, Youtube, Prime Video). Some of these distinguishing characteristics are:

- *Ephemeral Video Streams*: VOD content, like movies and TV shows, once available on the site can be viewed at any time. A

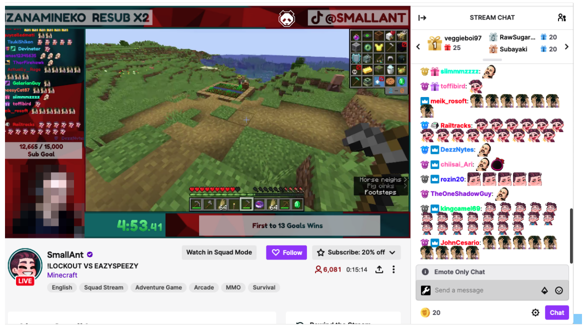

Figure 1: Channel page featuring streamer playing a game live. The streamer is blurred for privacy reasons. Right: Stream chat used by viewers to interact with the streamer.

Twitch live stream, on the other hand, is only available while live. The start time and duration of the live stream is decided by the individual streamer, and can vary anywhere from scheduled daily streams to completely irregular schedules or yearly bursts of activity, such as tournament streams. Thus, recommendations need to take into account the availability and duration/schedule of the streams for good performance.

- *Interactive Media*: Movies and TV shows only have video content. Twitch has live chat sessions showing right alongside the stream, where viewers not only interact with other viewers/streamer but can also with the streamer during the livestream. Community can be the primary reason for many Twitch viewers to join a stream.
- *Rapidly Varying Content*: Movies and TV shows have fixed and well-defined metadata about content, e.g. director, actors, genre etc. Detailed metadata is available through multiple standardized sources (e.g. IMDB), and remains mostly constant after production. For a Twitch live stream, the only fixed entity is the streamer. The content can change wildly over broadcasts and even midstream, e.g., at the start of a stream the streamer might be playing the game League of Legends and then switch to a different game, such as Fortnite later on. Therefore, recommendation system need to be aware of and respond to this rapidly updating metadata.
- *High streamer Inflow/Outflow*: Movies and TV shows have a high barrier to entry, while on Twitch there is a lower barrier to entry and a high rate of inflow and outflow of streamers. Thus, there is a significant need to make recommendations for streamers with minimum history and metadata, as well as the need to rapidly update recommendation models to deal with incoming and outgoing streamers.

At Twitch, we have built a recommendation system that benefit from adaptations which adjust and react to these characteristics. Next, we give an overview of the system (Section 1.1), and discuss two specific problems that arise in the live-streaming setting: availability and consumption measurement (Section 1.2). Following that, we go over related work (Section 2), describe our solutions to those two problems (Section 3), present experiment results (Section 4), and discuss potential improvements to data collection and evaluation based on our findings (Section 5).

1.1 Twitch's Recommendation System

In this section, we give an overview of the Twitch recommendation system. We describe the base model for which we make the loss modifications, how we perform model training, inference, and ultimately serve recommendations to our users.

1.1.1 *Model.*

To address the Twitch recommendations problem, we built the Channel Affinity Model (CAM), shown in Figure 2. It is a deep neural network that is trained on user watch history data. Each user is treated as a temporally weighted "bag of channels", where each channel is encoded as a trainable vector embedding. The resulting user representation is fed to a feedforward neural network. The resulting output is called the user encoding. We then take the dot product with the channel embeddings, which have the same weights as the input channel embeddings, to get the user-channel score. These scores are treated as the logits of a sigmoid function and trained to predict what the user will watch on the next day.

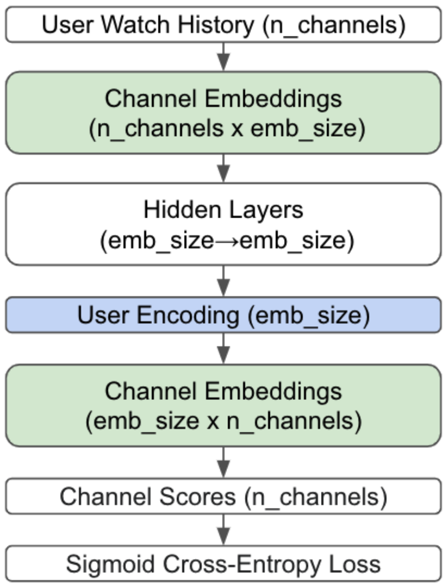
User Watch History (n_channels)
Channel Embeddings (n_channels x emb_size)
Hidden Layers (emb_size→emb_size)
User Encoding (emb_size)
Channel Embeddings (emb_size x n_channels)
Channel Scores (n_channels)
Sigmoid Cross-Entropy Loss

Figure 2: Diagram of the Channel Affinity Model. Note that user watch histories are represented as bag of channel embedding representations. Those same embeddings are used for a dot product based logits calculation and fed to one separate sigmoid function for each channel.

1.1.2 *Inference and Serving.*

When a user loads a page on Twitch, a request is made to the recommendations service, which looks up the user's CAM embedding and the CAM embeddings of all channels which are live. Notably, calculation of the final user-channel affinity scores is deferred to this stage, since we only need to compute scores for channels that are currently streaming. Since the service handles thousands of requests per second, calculating the dot product score for only available channels saves valuable latency and boosts throughput. The recommendations are then sorted and passed through various re-rankers and filters, which perform impression discounting [15] on the recommended channels, or filter into category specific surfaces that help users find Art, Music, and specific game streams.

1.2 Biases in Live Recommendation Systems

As mentioned above, in CAM, the preference of a user for a given channel is modeled as a separate scalar score for each user-channel pair. For the model training purposes, we employ watched channels as an implicit positive signal and non-watched channels as an implicit negative signal. While the positive signal is observable, assuming a non-watched channel as a negative signal can have undesired consequences on recommendations. For example, if a channel is offline when a user visits Twitch, assuming that as a negative signal does not capture the user's real preference. Furthermore, assuming that all positive signals have equal weight can also be detrimental to the recommendations model.

Availability: The issue of channel availability is unique to the Twitch live-streaming recommendations problem. While other video recommenders (e.g. Netflix, YouTube) have all content available for serving to the user at any time, only a subset of Twitch channels are available to be watched by a user who opens the site. The majority of channels are online for less than 4 hours a day, which means that users can and often do land on the site when their favorite streamer is taking a break, has not started streaming yet, or is already done streaming for the day. We currently measure a user's engagement by observing minutes watched on each channel by the user. Our goal is to infer the preference of a user, e.g., if they would prefer to watch channel i over all other channels, then train a model on these targets. However, the dynamic inventory of channels can cause unwarranted negative feedback that does not reflect the preferences of users. That is, if channel i is not online, then the user might watch channel j as a substitute, even though they would have preferred channel i. The set of available choices on any given day or any given time is limited, so minutes are biased toward channels with higher uptime.
Watch Time: Before the contributions mentioned here, we treated all video plays longer than the threshold of 5 minutes as equally important targets in training. Here, we want to measure the strength of a user's preference for a channel to assist in our ability to rank channels. In a fixed length video setting, such as YouTube or Netflix, one could measure the strength by calculating the completion percentage of a video. We cannot measure this information for a livestream, as they can often run for 4+ hours making it infeasible for users to watch an entire stream from start to finish. One correlated measure is the count of the number of minutes watched; for example, watching channel i for 2 hours likely indicates a stronger presence than watching channel j for 5 minutes. There are issues with measuring minutes as a proxy for true engagement, such as a channel being available for only 5 minutes after a user opens Twitch, or users leaving streams playing while not actively watching. However, being able to incorporate any information - even if it is imperfect - about preference strength is important and can improve the quality of the recommender.

To address these sources of bias in recommendations, in this paper, we propose two methods of weighting training loss during model training. The first method addresses the bias related to availability of channels, while the second method adjusts for the preferences implied by differing amounts of minutes watched.

2 RELATED WORK

Streaming Services. Live streaming content has grown substantially in popularity over the years, and Twitch has been studied for the social dynamics that arise from the interactions between communities and streamers. Hamilton, et al. [8] established that over time, communities can emerge from stream channels, through the participants' shared interests, experiences, and interactions, which encourages viewers to come back to the same streams over and over again. Deng et al. [5] find that the popularity of streams fluctuates greatly with a highly dynamic ecosystem of games and their popularity, with events like new releases and tournaments causing shifts in global viewership patterns. Gros et al. [7] study the various motivations of Twitch viewers, whether it is to follow tournaments, catch up with friends in chat, or learn new gaming strategies. Hilbert-Bruce et al. [9] find that viewers of smaller streams are more interested in social engagement. Recommender systems built for live streaming services can draw from this body of work to explore patterns that can inspire new techniques to improve models. One such pattern that we explore in this paper is irregular event streams which have low availability but high popularity when they are live; adjusting the loss function can help models value those streams more correctly.

Recommender Systems. Recommender systems model the interactions and relationships between users and items. Netflix found early success with matrix factorization techniques for VOD recommendations [13]. More recently, neural networks have been shown to be effective for recommendations on large-scale industrial datasets such as Prime Video [20] or YouTube [3]. More complex architectures have been developed to model temporal patterns in user behavior, such as recurrent neural networks [16], self-attentive networks [11], or transformer models [23]. Similarly, graph networks for recommendations have been developed to model the social interactions between users and communities [6], and have seen success at Pinterest [24]. Our work describes our modifications to the Twitch recommendation system which is large-scale and able to serve recommendations in production to millions of users.

Implicit Feedback. Recommenders can be trained and evaluated on either implicit or explicit feedback. At Twitch, we don't collect any explicit feedback, so we use implicit feedback as the basis of our recommender. Implicit feedback data is useful because we can easily collect a large amount of it to train large, data-hungry neural network models. However, it is subject to many issues, such as low data quality, relative frame of reference, and the inability to measure negative preferences. [10] The relative frame of reference can make it difficult to determine how much more positive one interaction is over another. One way to solidify the frame of reference that is similar to the method we propose is modeling expected watch time by weighting positive examples equal to the number of minutes watched. [3] There is no natural negative feedback in implicit feedback data, although there are methods which make assumptions about negative implicit feedback that benefit models. [17] The inability to precisely measure negative preferences can be exacerbated by the missing not-at-random problem [22], where user-item pairs which could have resulted in a positive interaction have no data due to presentation bias or other types of biases such as availability bias. Correcting biased training data can be achieved by using causal inference techniques such as inverse propensity weighting in recommender systems. [21] In practice, in large-scale recommender systems we have seen some applications of inverse propensity weighting to counteract position bias. [2] [18] Recently, there has been work on counteracting availability bias on Twitch data by using self-attention over only relevant and available items to simultaneously calculate temporal embeddings for that set of items. [19] Our work addresses how we counteract availability and consumption biases that arise from the way that we collect implicit feedback data in the form of counting video plays in watch history. Our method also differs from [19] in that ours performs batch recommendations for live channels, while theirs is trained as a "next item to watch" recommender.

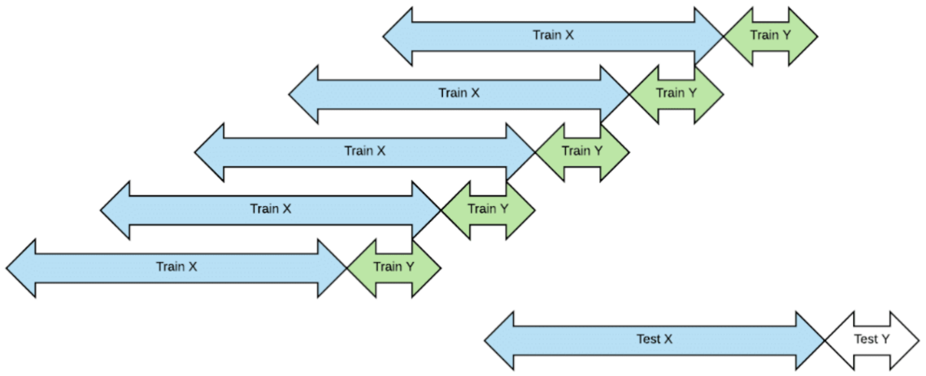

Figure 3: Data augmentation method showing the splitting of training/validation input and output into slices. Each slice is identical in form (same length of input and output data as the validation set) but has different target evaluation days, allowing us to train on a higher number of data samples per user.

3 METHOD

3.1 Problem Statement

We formulate the recommendation problem as a pointwise learning-to-rank task. That is, given recent user history, calculate user-channel scores $s_{u,i}, s_{u,j}$ individually for each user-channel pair $(u, i), (u, j)$. Afterwards, we sort the scores and recommend live channels with the highest scores for a given user u.

CAM is a feedforward neural network that outputs the probability, $p(y_{u,i})$, of a user watching a live channel. For the purposes of input, each user u is treated as a "bag of channels", C_u, based on the channels they watched in the last N days.

Channels are embedded as as n-dimensional vectors e_i, and we take a temporally weighted sum of these embeddings and feed them through a 3-layer residual feedforward neural network. This results in an n-dimensional user encoding e_u. We then take the dot product with each channel embedding e_i to obtain a scalar channel score for each user, represented as $s_{u,i} = e_u^\top e_i$. Next we estimate the probability of the user u watching the channel i on the target day as $p(y_{u,i}) = \sigma(s_{u,i})$. Separately, we calculate the target $y_{u,i}$, which is computed from the number of minutes of channel i watched by user u, notated as $m_{u,i}$. We treat all video plays longer than a threshold of 3 to 10 minutes as important targets. We optimize the model using a mixture of binary cross entropy loss functions:

$$L_u = \sum_i^C y_{u,i} \log(p(y_{u,i})) + (1 - y_{u,i}) \log(1 - p(y_{u,i})) \tag{1}$$

Our dataset uses as input the last N days of minutes watched data for a user, and outputs the predicted probability of the user watching each channel on the next day. We create M overlapping slices of the data per user with different target days and input data from each user, offsetting each data slice by an increasing number of days as shown in Figure 3. This is similar to a Time-delayed neural network [14] where the different time steps are combined with a fixed set of temporally decaying weights.

3.2 Availability Weighting

To address the issue of availability, we propose a modification of the loss function. For each target day, we calculate the fraction of the day which each channel was online as a_i, and use it as a weight on the negative samples only. We use the following loss function variant:

$$L_u = \sum_i^C y_{u,i} \log(p(y_{u,i})) + \boldsymbol{a_i}(1 - y_{u,i} \log(1 - p(y_{u,i})) \tag{2}$$

This means that if channel i does not stream on the target day, it has $a_i = 0$, which means that all negative examples do not count against it. Conversely, a stream that has high uptime on the target day will have high a_i, which means that negative examples will count more significantly against it, since more users had a chance to watch the stream for a longer period of time.

This method is not a perfect approximation of the actual effect of a channel being offline to a user, as it assumes that users watch Twitch at a uniformly random time in each 24 hour window. Hypothetically, a channel i which only broadcasts for 6 hours when a user is asleep is never available to that user and should have $a_i = 0$ for that particular user, but in the current scheme it would have $a_i = 0.25$. We use this approximation because it is computationally too expensive to calculate every triplet of interactions of users and channels being online at time t of the day for the full set of users and channels we consider at scale.

3.3 Minutes Weighting

To effectively use the number of minutes watched as a measure of significance of preference, we propose another modification of the loss function. We weight each positive example by a function f_m of the number of minutes watched. In our experiments, we try using the identity function and $f_m(m_{u,i}) = \log_2(m_{u,i})$. Both functions monotonically increase the weight of positive samples when we observe increased minutes watched.

$$L_u = \sum_i^C \boldsymbol{f_m(m_{u,i})} y_{u,i} \log(p(y_{u,i})) + (1 - y_{u,i}) \log(1 - p(y_{u,i})) \tag{3}$$

Table 1: Minutes offline metrics, measured relative to Baseline

Model	P@5	R@50	CIC@50
Baseline	**0.00%**	0.00%	0.00%
$f_m(m) = m$	-3.72%	-0.58%	1.36%
$f_m(m) = \log_2(m)$	-0.65%	**0.09%**	**2.35%**

This method does not adjust for availability; for example, channels with a short broadcast on a target day have a cap on the number of minutes that users can watch. The relationship between minutes watched, user and channel availability schedules, presentation bias, and user preference strength is complicated; for the purposes of this work we assume that minutes are a good enough approximation for the user preference strength.

To form the final loss function, we combine both weighting schemes into one final loss function.

$$L_u = \sum_{i}^{C} \boldsymbol{f_m(m_{u,i})} y_{u,i} \log(p(y_{u,i})) + \boldsymbol{a_i}(1 - y_{u,i}) \log(1 - p(y_{u,i})) \tag{4}$$

4 EXPERIMENTS

We train the model described in Section 3.1 and Figure 2 on $N = 35$ days of historical data divided into $M = 7$ slices per user using the multiple window scheme shown in Figure 3. We use an embedding size of $n = 256$ for the channel embeddings and user encodings, and 3 hidden layers of size 256 for the transformation from bag of channels to user encoding. We use the loss function in Equation 1 as the baseline, Equation 3 as the Minutes variant, and Equation 4 as the Availability variant which builds on the Minutes variant. To optimize the model, we use the Adam [12] optimizer with learning rate 0.005.

4.1 Offline Evaluation

We then evaluate the trained models offline by running inference on the left-out following day and comparing against the ground truth of what users actually watched on that day. The channels are sorted in order of highest predicted watch probability to form the list of recommendations for each user. We then calculate the precision P of recommendations made by the model, scoring 1 if the model predicts that a user watches a channel for at least a few minutes in the top $k = 5$ predictions, and 0 otherwise. We calculate the recall R at $k = 50$ similarly. We also calculate converted item coverage CIC, which is a metric that measures the percentage of channels out of the total recommendable channel pool which are successfully recommended to users.

Minutes: Table 1 shows the results of the model trained with just the minutes weighting applied to positive examples (Equation 3). The results show that while weighting examples directly with the minutes doesn't improve performance at all, when we apply a log function to the weights, we observe a 0.7% decrease in precision@5, a 0.1% increase in recall@50, and a 2.4% increase in converted item coverage.

Table 2: Availability offline metrics, measured relative to Baseline

Model	P@5	R@50	CIC@50
Baseline	**0.00%**	**0.00%**	0.00%
Avail	-9.62%	-1.22%	**1.66%**

Table 3: Availability adjusted offline metrics, measured relative to Baseline

Model	P@5	R@50	CIC@50
Baseline	0.00%	0.00%	0.00%
Avail	**0.93%**	**0.45%**	**2.70%**

The number of minutes watched is ignored in the metrics, so if the model predicts one channel that a user watched heavily, it's less significant than predicting multiple channels that a user browsed through quickly. We can adjust for this by weighting the evaluation metrics by the number of minutes watched, to calculate recall of minutes predicted instead of recall of watches predicted.

The minute-weighted model showed a decrease of 0.2% in minute-weighted recall at $k = 5$. Surprisingly, the minute-weighted model didn't improve the minute-weighted recall at all, despite having a more accurate optimization target for the new metric.

Availability: Table 2 shows the results of the model trained with the full modified loss function (Equation 4), including both up-weighting losses for longer watches, and downweighting losses for low-availability channels. The baseline in this case is the minutes weighted model trained on Equation 3. We note that the offline performance of the model is actually worse than baseline in all aspects besides converted item coverage. These effects are likely due to the increase of successful recommendations of channels with low availability that are underrepresented in the input dataset and original loss function.

In a real scenario, these channels would have been recommended highly for the small number of hours they were online, and the recommendation slots would have been filled by more highly available channels the rest of the day. In our offline evaluation, however, the highly recommended slot is given to the low availability channel, which results in less successful conversions. It's difficult to make adjustments to the evaluation function to adjust for this bias, since it is hard to replicate the set of available choices to a user through every moment of the day due to infrastructure limitations.

As a partial correction for this effect, we tried adjusting the metrics to reflect availability by removing channels that were not available on the evaluation day from the pool of recommendable channels. This is not perfect, since channels still have varying availability through the different hours of the day, but we observe that the results in Table 3 show that the availability-adjusted model is better than the non-adjusted model in all metrics, increasing precision@5 by 0.8%.

4.2 Online Evaluation

To evaluate our models online, we run an A/B test against an established baseline model on our recommendation surfaces. We measure the number of minutes watched that occur when a user clicks on a recommendation provided by a model on a surface as recommended minutes. We also measure the total number of minutes watched sitewide. Since we only show recommendations for live channels, our online evaluation adjusts for availability issues correctly. Similarly, since we count the number of minutes, we adjust for the preferences potentially shown by spending more minutes watching specific channels. We ran the A/B tests for 2 weeks assigning users randomly to control and treatment groups. Statistical significance tests are calculated using a standard control variate model with minute-watched differences from before and during treatment as surrogate.

We tested the minute-weighted model against the baseline, which did not have weighting enabled. This experiment was a success, increasing the minutes generated by the recommender by 1.3%.

We tested the availability-weighted model against the baseline, which already had minutes weighting enabled. This was the largest single improvement in a deep learning model A/B test in Twitch recommendations history, increasing the number of recommended minutes generated by the recommender by 6.5%. On top of having a large lift on minutes referred through our model, the model also produced a significant total minutes lift of 0.22%. Overall, the two experiments together contributed 7.9% improvement to our recommended minutes.

5 DISCUSSIONS AND FUTURE WORK

In this section, we will discuss some interesting findings around strong online performance gains that were led by weak offline metrics, which suggests our offline evaluation can use improvement (5.1). Then we will discuss extensions of our work to further our understanding of users' strength of preference (5.2) and the availability interactions between users and channels (5.3).

5.1 Correlation of Offline Metrics to Online Performance

For both of the methods we discussed, the original offline results were mixed or slightly negative. Even after weighting our metrics to be aligned with the loss weighting methods, we saw a small improvement for availability weighting and no improvement for minutes weighting. All of these offline results were failed to predict the actual online A/B results that we gathered by testing the models on users, which showed that both weighting methods were very successful in making better recommendations. This suggests that our original metrics were not correlating well with the online performance of models. This speaks to the difficulty of picking good metrics for our recommendations surfaces, and recommender systems in general.

We decided to incorporate the availability adjusted evaluation method described in Section 4.1 into our evaluation metrics for future offline testing. However, that method was just a simple approximation of availability, and future work could improve on the assumptions by better measuring channel availability against users' availabilities in evaluation. Future work could also find a better way to express the minutes-based preference of users in a way that better correlates with online performance of models.

5.2 Exploring More Methods for Measuring Significance of Preference

Beyond watching, there are many other activities that viewers on Twitch can partake in that can indicate significance of preference. Activities such as following, subscribing and donating bits (which are two methods of channel patronage), and chatting in a channel can all indicate increased preference for a channel, compared to a channel for which a user takes no actions. Future research could find ways to leverage these other forms of user-channel interaction as input or output data, as well as using them as additional tasks in multi-task learning.

5.3 Better Methods for Measuring Availability

As we discussed in Section 3.2, the current method of measuring availability is not ideal. We do not compare a granular view of each user and channel's available times, so a channel which is only available when a user is asleep will still get a negative score for that user. We also do not measure the interaction effect of multiple channels being available at the same time– one channel might be preferred by a user, but another substitute channel may be almost as important to the user. It is technically difficult to measure the user-channel availability at such a granular scale for all user-channel pairs, but it may be possible to incorporate this information into a second-stage model which analyzes this data for a fraction of user-channel pairs. Another data source could be the set of channels which made an impression on a user through the various surfaces on Twitch, that would illuminate which items were actually shown to the user during on-site browsing.

6 CONCLUSION

It is simpler to assume that data is i.i.d and to apply standard supervised learning techniques to create a working product. But in recommendation systems in live marketplaces such as Twitch, where both inventory and users' tastes are dynamic, such simplifying assumptions can be suboptimal. We showed that models that try to explicitly incorporate such dynamics can be beneficial. In this work, this explicit modeling took form in reweighting schemes of the binary cross entropy loss functions. Both consumption level (minute-watched weighting) and causality (availability weighting) inspired weights improved our metrics in online A/B tests.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 265–283.

[2] Alexey Borisov, Ilya Markov, Maarten de Rijke, and Pavel Serdyukov. 2016. A Neural Click Model for Web Search. In *Proceedings of the 25th International Conference on World Wide Web* (Montréal, Québec, Canada). International World Wide Web Conferences Steering Committee, 531–541. https://doi.org/10.1145/2872427.2883033

[3] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198. https://dl.acm.org/doi/pdf/10.1145/2959100.2959190

[4] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall,

and Werner Vogels. 2007. Dynamo: Amazon's highly available key-value store. *ACM SIGOPS operating systems review* 41, 6 (2007), 205–220.

[5] Jie Deng, Felix Cuadrado, Gareth Tyson, and Steve Uhlig. 2015. Behind the game: Exploring the twitch streaming platform. In *2015 International Workshop on Network and Systems Support for Games (NetGames)*. 1–6. https://doi.org/10.1109/NetGames.2015.7382994

[6] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *The World Wide Web Conference*. 417–426.

[7] Daniel Gros, Brigitta Wanner, Anna Hackenholt, Piotr Zawadzki, and Kathrin Knautz. 2017. World of Streaming. Motivation and Gratification on Twitch. In *Social Computing and Social Media. Human Behavior*, Gabriele Meiselwitz (Ed.). Springer International Publishing, Cham, 44–57.

[8] William A. Hamilton, Oliver Garretson, and Andruid Kerne. 2014. Streaming on Twitch: Fostering Participatory Communities of Play within Live Mixed Media. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 1315–1324. https://doi.org/10.1145/2556288.2557048

[9] Zorah Hilvert-Bruce, James T. Neill, Max Sjöblom, and Juho Hamari. 2018. Social motivations of live-streaming viewer engagement on Twitch. *Computers in Human Behavior* 84 (2018), 58–67. https://doi.org/10.1016/j.chb.2018.02.013

[10] Gawesh Jawaheer, Martin Szomszor, and Patty Kostkova. 2010. Comparison of Implicit and Explicit Feedback from an Online Music Recommendation Service. In *Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems* (Barcelona, Spain) *(HetRec '10)*. Association for Computing Machinery, New York, NY, USA, 47–51. https://doi.org/10.1145/1869446.1869453

[11] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 197–206.

[12] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[13] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[14] Kevin J. Lang, Alex H. Waibel, and Geoffrey E. Hinton. 1990. A time-delay neural network architecture for isolated word recognition. *Neural Networks* 3, 1 (1990), 23–43. https://www.sciencedirect.com/science/article/pii/089360809090044L

[15] Pei Lee, Laks V.S. Lakshmanan, Mitul Tiwari, and Sam Shah. 2014. Modeling Impression Discounting in Large-Scale Recommender Systems. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, New York, USA) *(KDD '14)*. Association for Computing Machinery, New York, NY, USA, 1837–1846. https://doi.org/10.1145/2623330.2623356

[16] Qiang Liu, Shu Wu, Diyi Wang, Zhaokang Li, and Liang Wang. 2016. Context-aware sequential recommendation. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 1053–1058.

[17] Ladislav Peska and Peter Vojtas. 2013. Negative implicit feedback in e-commerce recommender systems. In *Proceedings of the 3rd International Conference on Web Intelligence, Mining and Semantics*. 1–4.

[18] Zhen Qin, Suming J Chen, Donald Metzler, Yongwoo Noh, Jingzheng Qin, and Xuanhui Wang. 2020. Attribute-based propensity for unbiased learning in recommender systems: Algorithm and case studies. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2359–2367.

[19] Jérémie Rappaz, Julian McAuley, and Karl Aberer. 2021. Recommendation on Live-Streaming Platforms: Dynamic Availability and Repeat Consumption. In *RecSys '21: Fifteenth ACM Conference on Recommender Systems* (Amsterdam, Netherlands). https://cseweb.ucsd.edu/~jmcauley/pdfs/recsys21b.pdf

[20] Oleg Rybakov, Vijai Mohan, Avishkar Misra, Scott LeGrand, Rejith Joseph, Kiuk Chung, Siddharth Singh, Qian You, Eric T. Nalisnick, Leo Dirac, and Runfei Luo. 2018. The Effectiveness of a two-Layer Neural Network for Recommendations. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=S108WDJvf

[21] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. 2016. Recommendations as treatments: Debiasing learning and evaluation. In *international conference on machine learning*. PMLR, 1670–1679.

[22] Harald Steck. 2010. Training and Testing of Recommender Systems on Data Missing Not at Random. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA, 713–722. https://doi.org/10.1145/1835804.1835895

[23] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 1441–1450.

[24] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.