Managing Popularity Bias in Session-Based Recommendation

Flavien Vidal

École Polytechnique
flavien.vidal@polytechnique.edu
June 2022

Abstract

Back in 2006, Netflix hosted a competition to solve a problem that had been well known for years [10]. All the tech giants were already struggling to solve this very same problem: given a user's past behavior, how can we predict what that same user will like in the future? This is the question that recommender systems try to answer. Since then, the field of online recommendation has dramatically evolved. Anyone shopping on Amazon, listening to music on Spotify or searching for TV shows on Netflix is experiencing personalized recommendations. There are a plethora of methods for providing these recommendations to users but one of their limitations is the problem of popularity bias, which surprisingly receives very little attention. Among the various models, session-based recommendation which deals with anonymous session data is particularly affected by this bias. The development of a new method capable of addressing these difficulties is therefore of great importance. In this work, we introduce a comprehensive and flexible approach to balance the accuracy and coverage of recommendations. To the best of our knowledge, studies conducted on session-based long-tail recommendation were based on artificial data and did not offer deep analyses (quantitative and qualitative studies) of the effects of their approaches on performance (not only in terms of precision and recall but also by evaluating a variety of popularity metrics). The main contributions of this work are threefold. First, we present a regularization-based framework and show that it provides an adjustable mechanism for controlling the balance between precision and coverage. Second, we propose an attention mechanism that determines the user's preference between long-tail and short-head items in order to adjust recommendations. Finally, we conduct extensive experiments on real-world datasets that fluctuate from day to day and are directly affected by the counterfactual feedback loop.

Definitions and Abbreviations

Popularity: there are many definitions in the literature, all specific to particular uses. For example, in the context of videos, it is common to define the popularity of a video as either the number of views it generates, the number of clicks it generates, the total viewing time it generates, or the average viewing percentage it generates. All definitions have their advantages and disadvantages, but there is no universal definition of popularity.

Short-head item: item belonging to the short-head of the popularity distribution according to the Pareto principle.

Long-tail item: item belonging to the long-tail of the popularity distribution according to the Pareto principle.

Popularity bias: it is a well-known phenomenon in recommender systems where popular items are recommended even more frequently than their popularity would warrant, amplifying long-tail effects already present in many recommendation domains [1].

Novelty: this term has different definitions in the literature. A novel item may refer to a recently added item users have not yet evaluated [9] (typically during the cold start problem [14]) but most often, novelty refers to how different an item is with respect to "what has been previously seen" by a specific user or by a community as a whole.

Diveristy: it is a different but related concept from novelty. Diversity generally applies to a set of items, and is related to how different the items are with respect to each other. This is related to novelty in that when a set is diverse, each item is "novel" with respect to the rest of the set.

Serendipity: it is a complex concept related to relevance, novelty and unexpectedness. To improve user satisfaction, recommender systems should offer serendipitous suggestions: items not only relevant and novel to the target user, but also significantly different from the items that the user has rated. However, the concept of serendipity is very subjective and serendipitous encounters are very rare in real-world scenarios, which makes serendipitous recommendations extremely difficult to study. To date, various definitions and evaluation metrics to measure serendipity have been proposed, and there is no wide consensus on which definition and evaluation metric to use.

Session: a session consists of an ordered sequence of events.

Event: it is used to describe a user's click and is composed of several categorical variables: video, owner and channel identifiers, topics (first and second) and their root classes, and finally IAB categories (first and second).

Introduction

Whilst recommendation has their own dedicated conference (ACM Recommender Systems Conference), it is not the most popular topic in the academic world and it receives little attention at scientific conferences compared to other fields of machine learning. However, improving recommendations is a strataegic challenge for all companies selling goods or services online. Better recommendation turns into more revenues, higher margins and stronger customer engagement.

Historically recommender systems have tried to build models based on user profiles. In this context, factoring methods and neighborhood models have dominated the literature. Should a user's profile not be available, cookies and browser fingerprints can provide some level of user recognition but these technologies are often not reliable enough. The challenge is that it requires a lot of data hence it becomes less relevant for users making few sessions on websites. Beyond that, the behavior of a user often exhibits session-related characteristics because his tastes evolve over time. In order to take into account his short-term preferences, his subsequent sessions can therefore be processed independently.

Session-based recommendation provides a solution when user profiles are not available. It aims at predicting user behaviors based on anonymous sessions, either for privacy reasons or simply because users are not logged in. These recommenders only rely on large, time-ordered action logs of anonymous users to predict the user's next action [11]. Their limitation lies in the fact that they are based on relatively simple methods that often only take into account the last click of the user, without taking into account information about previous clicks (e.g., similarity between items, co-occurrence or transition probabilities).

The last decade has seen the tremendous success of deep learning in natural language processing with conversation modeling or text generation. To model the sequential data of these problems, recurrent neural networks have undeniably been the model of choice and as a result of this shift, session-based recommendation has received much more attention than ever. Indeed, session-based recommendation shares some similarities with NLP problems in that they both deal with sequential data. For example, we can consider the first item the user clicks on as the first entry in the RNN, and then try to predict the next most likely item. Each subsequent click by the user will be the next entry in our network. In 2016, Hidasi et al. [7] adapted RNNs to the recommendation context with remarkable results.

However little attention has been paid to the challenge of popularity bias [5]: collaborative filtering methods generally focus on popular (short-head) items at the expense of niche (long-tail) ones [13]. While these popular items are often good recommendations, it does not promote the discovery of new items and even ignores the interests of some users with specific tastes. Despite this finding, most studies focus primarily on achieving maximum accuracy, ignoring long-tail recommendations, which nevertheless play an important role in recommendation diversity and serendipity. Since long-tail distributions are common in recommendation scenarios, and even more so in session-based recommendation, more attention should be paid to popularity bias.

Previous studies provide more complete explanations of the importance of long tail recommendation. First, from the users' point of view, recommending only popular items can quickly become annoying, while recommending niche items promotes diversity and serendipity, allowing for user surprise and satisfaction [3]. Secondly, from the producers' point of view, systematically recommending popular items could lead to their niche or newer products being ignored [2]. Finally, from a business perspective, niche products may enable higher pricing and benefits compared to popular products [17].

Many obstacles such as data sparsity, cold start or counterfactual feedback loop stand in the way of applying long-tail recommendation, resulting in most existing recommendation systems being biased towards popular items. In these methods, the preference to recommend the most popular items when the model is uncertain is a conservative but effective way to improve accuracy [3]. Previous works focusing on long-tail recommendations either sacrifice the accuracy of recommendations [3] [8] [15], or adopt side-information (such as user profiles), which exceeds the data limit of session-based recommendation, to mitigate long-tail items data sparsity.

The complexity of long-tail recommendation is even greater for session-based models. First, the lack of secondary information in sessions prevents the application of traditional long-tail recommendation methods. Second, the sequential aspect of the data does not allow the application of these same methods. Third, session data is particularly sparse because each session (even from the same user) must be processed independently [6].

For all these reasons, the development of a new method that can solve the above difficulties is of great significance. In this work, we address the topic of popularity bias in session-based recommendation with RNNs and try to provide a general and flexible approach to balance between accuracy and novelty of recommendations. First, we introduce the general architecture of the recommendation module, the data used and the evaluation metrics specific to this project. Then, we present a regularization-based framework to enhance the long-tail coverage of recommendation lists and show that it constitutes a tunable mechanism to control the balance of recommendations. Finally, we introduce a preference mechanism that determines the user's preference between niche and popular items in order to adjust the recommendation mode: recommending more short-head or long-tail items. Our approach provides an easy way to tune a recommendation model to achieve the desired tradeoff between accuracy and novelty. Unlike the majority of session-based models, our work improves the recovery of long-tail items while maintaining consistent performance simultaneously.

To our knowledge, and until now, studies conducted on the long-tail recommendation were based on artificial data. Our study is conducted on real-world data that fluctuates from day to day and is directly influenced by users and models in production and thus is affected by the counterfactual feedback loop. Moreover, no previous research analyzed in depth the effects of popularity regularization on performance in the context of session-based recommendation. Not only do we compare the effect of different regularizations in terms of precision and recall but also in terms of popularity metrics (average popularity, coverage, long-tail coverage, tail-

coverage, ...). In the light of these first quantitative results, we provide a qualitative study that illustrates what happens in practice when the regularization parameter varies and for example when the input items are in the distant-tail, long-tail and short-head.

1 Formulation of the problem

In this section we introduce the video-to-video (V2V) recommendation module. We start by presenting the architecture of our network. Then we present the approach we use to create the batches for the training step. Finally, we describe and discuss the data we collect.

1.1 Architecture of the network

Let $D_{rv} = \{v_1, v_2, \ldots, v_{|D_{rv}|}\}$ be the set of unique items in all sessions where rv stands for "recommendable videos". Let $s_i = \{v_{i,1}, v_{i,2}, \ldots, v_{i,t}\}$ represent a session consisting of t events ordered by timestamps, starting at event $v_{i,1}$ and ending at event $v_{i,t}$. An event is used to describe a user's click and is composed of several categorical variables that are presented in the first column of Table 1: video, owner and channel identifiers, topics (first and second) and their root classes, and finally IAB categories (first and second). The aim of the V2V recommendation module can be defined as using the user's current session data s_i to predict the user's next clicked item $v_{i,t+1}$. It is therefore a typical next-click prediction problem. We then define $e(s_i) = \{e(v_{i,1}), e(v_{i,2}), \ldots, e(v_{i,t})\}$ that represents the latent representation for the same session encoded by the embedding layer, where $e(v_{i,t}) \in e(s_i)$ is the latent representation of the event of session s_i at timestamp t. To derive embedding $e(v_{i,t})$, we first compute the embeddings related to each categorical variable independently and concatenate them in order to form a single embedding gathering all the information contained by event $v_{i,t}$:

$$e(v_{i,t}) = [e(video_{i,t}),\ e(owner_{i,t}),\ e(topic_{i,t}),\ e(channel_{i,t}),\ e(iab_{i,t})] \tag{1}$$

Once evaluated, these emebeddings are given as input to the core of our network which consists of a recurrent neural network (RNN). To circumvent the issue of vanishing gradient we use gated recurrent units (GRU) [6] where the dependence structure between the forget and input gates is explicitly specified. The forget gate is set to one minus the writing gate such that:

$$\begin{aligned} h_t &= f_t \odot h_{t-1} + i_t \odot \tilde{h}_t \\ &= f_t \odot h_{t-1} + (1 - f_t) \odot \tilde{h}_t \end{aligned} \tag{2}$$

In that case, the new hidden state h_t is a weighted average of the previous hidden state h_{t-1} and the newly created candidate $\tilde{h}_t$. Consequently, h_t is bounded if h_{t-1} and $\tilde{h}_t$ are, which is the case using bounded activation functions. The GRU cell we use is therefore defined as follow:

$$o_t = \sigma(W_{o,h} \cdot h_{t-1} + W_{o,x} \cdot e(v_{i,t}) + b_o) \tag{3}$$

$$f_t = \sigma(W_{f,h} \cdot h_{t-1} + W_{f,x} \cdot e(v_{i,t}) + b_f) \tag{4}$$

$$\tilde{h}_t = \tanh(W_h \cdot (o_t \odot h_{t-1}) + W_x \cdot e(v_{i,t}) + b \tag{5}$$

$$h_t = f_t \odot h_{t-1} + (1 - f_t) \odot \tilde{h}_t \tag{6}$$

where o_t and f_t denote the output and forget gates, $W_{o,h}$ and $W_{f,h}$ are the corresponding weights, and σ denotes the sigmoid function. We initialize $h_0 = 0_d$.

The output h_t of the last GRU cell is then concatenated to an embedding $e(context_i)$ which corresponds to the concatenation of the embeddings of the remaining inputs presented in the second column of Table 1: the page, the country and platform of the visitor as well as the time since the last viewing:

$$e(context_i) = [e(page_{i,t}), e(country_{i,t}), e(platform_{i,t}), e(elapsed\_time_{i,t})] \tag{7}$$

The decision to use these input variables separately from the others is driven by the fact that they are shared throughout the session, whereas the previous ones vary from event to event.

A rectified linear unit (ReLU) non-linearity is applied element-wise to the latter vector before being given as input to a fully connected layer whose activation function is also a ReLU and whose output is the predicted embedding vector $\hat{e}(v_{i,t+1})$. The relevance scores $[relevance(i, s_i)]_{i \in D_{rv}}$ of recommending videos v_i for session s_i are computed as the similarity scores between the predicted vector $\hat{e}(v_{i,t+1})$ and the embedding vectors of the given videos $e(v_i, s_i)$. This is typically done by the scalar product or cosine similarity of vector $\hat{e}(v_{i,t+1})$ and each column vectors of matrix $U_{|D_{rv}| \times d}$ which holds the d-dimensional embeddings of the videos in catalog D_{rv}. These scores reflect the raw predictions of the network and are referred to as $\hat{y}_{raw}$.

During training, a softmax function is applied to this vector to generate a probability vector $\hat{y}$, where each component $\hat{y}_i$ is the probability that video v_i is the next video. During serving,vector $\hat{y}_{raw}$ is directly used as is in order to provide the user with the n videos associated with the highest scores.

The complete architecture is shown in Figure 1 which illustrates a session of size four in a time series of events. The objective here is therefore to determine the video $v_{i,5}$.

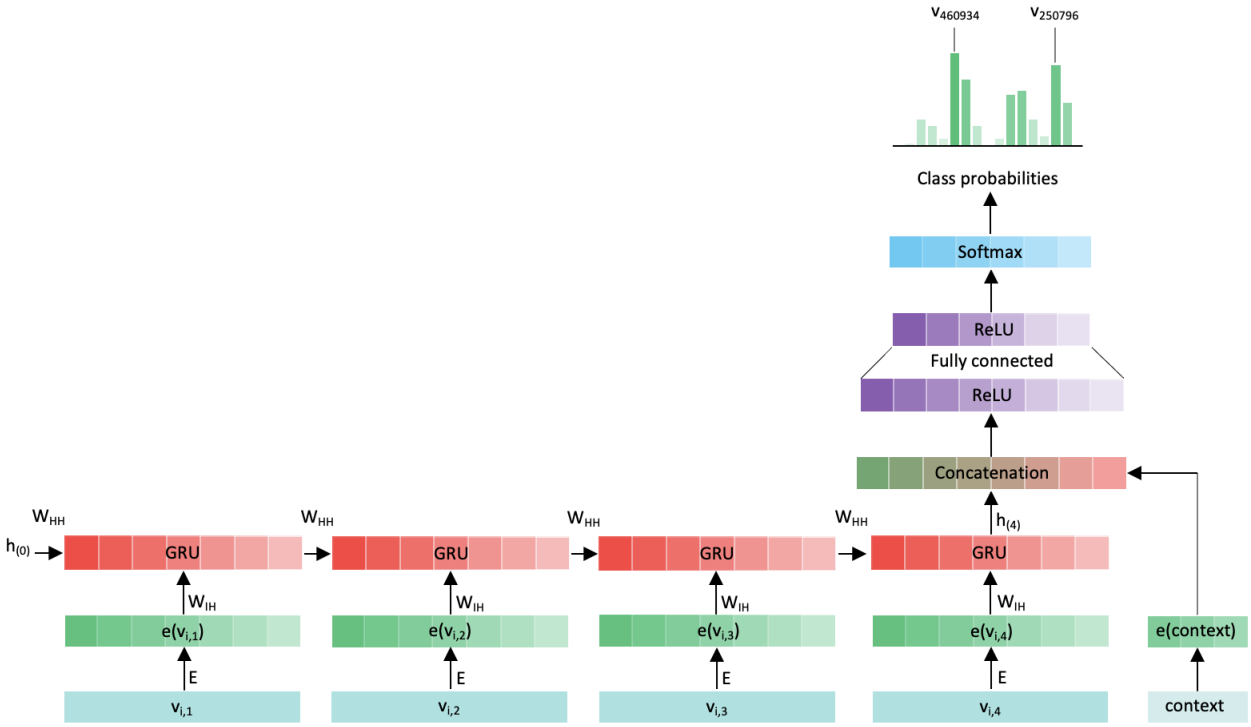


Figure 1: Architecture of the V2V recommender system

1.2 Session-parallel mini-batches

Recurrent neural networks used in natural language processing tasks typically use in-sequence mini-batches. For example, it is common to use a sliding window on words of sentences and place these slices next to each other to form mini-batches. This method is not suitable for our task. First, because the length of the sessions can be very different (more than the length of the sentences): some sessions have only 2 events while others can have hundreds. Secondly, because our goal is to capture the evolution of a session over time and decomposition into fragments would not make sense. For this reason, we choose to use session-parallel mini-batches. First, we create an order for the sessions. Then, the input of the first mini-batch is formed by the first event of the first m_b sessions while its output is formed by the second event of these same m_b sessions. The second mini-batch is then formed by the second events of the first m_b sessions and so on. When the last element of one of the sessions has been used as a target of a mini-batch, the next available session is considered instead. Since the sessions are assumed to be independent, the GRU hidden state is reset to zero at this session change. This process is illustrated on Figure 2.

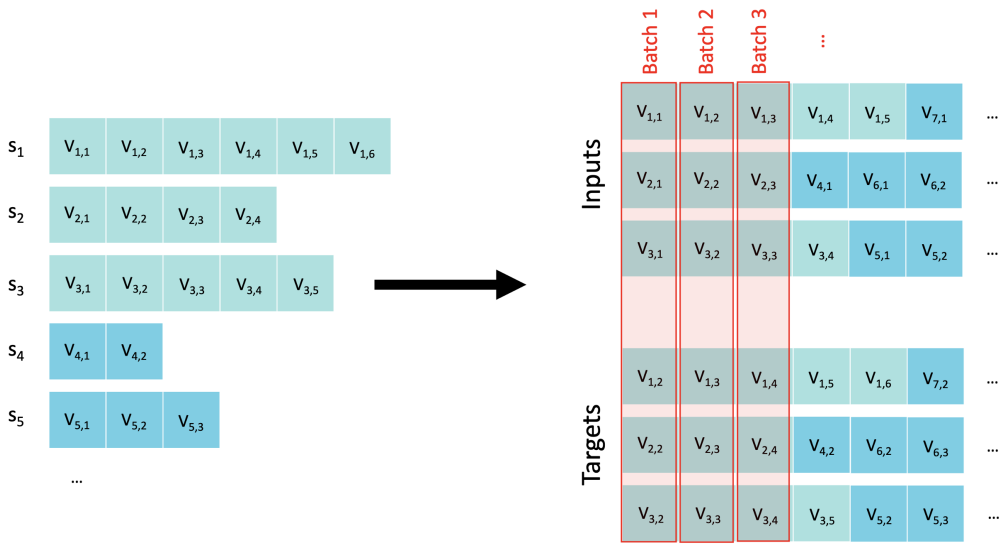


Figure 2: Session-parallel mini-batches

1.3 Collecting data

The recommendation field faces a major issue: catalogs of selectable items often contain thousands, hundreds of thousands, or even millions of items. In addition to the size of the catalogs, another challenge is that the click stream dataset is usually also very large. On its side, Dailymotion hosts hundreds of millions of videos that are watched by millions of users every day.

When training our models, only the data collected over the last 16 days are considered. This represents an average of 2.5 million different watched videos and nearly 200 million observations. In this work, only the French data are considered. In addition, we filter out very long sessions, as they are likely to have been generated by bots. We only work with some information from the clicks and in particular, we keep the variables listed in Table 1. Sessions collected the day after the 16 days are used for the validation step. Each session is assigned to either the training set or the test set, we do not split the data mid-session. Due to the nature of collaborative filtering methods, clicks corresponding to unlisted items are modified by default values that are adjusted through time.

Note that the data collected for training and evaluation are themselves influenced by the model in production.

In what follows, we evaluate the changes introduced with respect to the baseline that we defined in this section. Unless explicitly stated otherwise, our experiments are performed on different datasets obtained by considering different time windows in order to check the stability of our results.

Table 1: Input variables used in the V2V recommendation.

Input Variables	
Session[1]	Context[2]
video_id_session_i	time_since_last_view†
owner_id_session_i	visitor_country
root_class_first_topic_i	visitor_platform
root_class_second_topic_i	page
channel_id_session_i	
first_category_iab_i	
second_category_iab_i	
first_topic_i	
second_topic_i	

[1] We call session variables the click-level inputs that vary throughout the session.
[2] We call context variables the inputs that are common to all clicks in the session.
† indicates the variable is continuous.

1.4 Popularity bias

Before going deeper into the subject of popularity bias, we try to draw a definition of what popularity is, or at least of what it should be. There are many definitions in the literature, all of them specific to particular uses. For example, in the context of videos, popularity is often defined in one of these ways:

- The number of views it generates,
- The number of clicks it generates,
- The total watching time it generates,
- The average viewing percentage it generates.

All these definitions have their advantages and disadvantages but none of them work properly in all situations. There is no universal definition of popularity.

For example, if we consider the first definition, popularity accounts for videos that were not necessarily chosen by users, for example because of autoplay. To overcome these problems, we can consider the second definition. This time the videos accounted for have been chosen by the users, but we account for all clicks in the same way. If users click on a video and quickly realize, say after ten seconds, that the video does not interest them, is it legitimate to consider these clicks in the popularity computation? Many companies, including Netflix, use the third definition (total watching time) to overcome these difficulties. However, if we consider sixty users who watch a full one-minute video, and one last user who watches a full one-hour video, what can we conclude in terms of popularity? Are the two videos indeed as popular as each other? The fourth definition (average percentage of views) is also commonly

used, but in addition to being highly dependent on the number of views, it only makes sense when the set of videos being compared have similar and relatively high durations. Regardless of our interest, it is much more likely to watch a full one-minute video than a full one-hour video. This definition therefore strongly favors short videos. The age of the video is also a barrier. Regardless of the definition, an older, well-established video will tend to be more popular than a newer video.

Even though it may seem rather simple and intuitive, choosing a definition of popularity is often not as straightforward as one might imagine. However, this is an important step in modeling our problem and we need to agree on what we consider to be popular or niche before trying to boost some videos and penalize others.

In the remainder of our study, we define popularity as the average number of clicks a video has generated, and we set conditions to take the view into account (has the video been viewed more than x seconds, ...)

Having defined the popularity, we can visualize it in Figure 3 where coordinate axis represents the popularity of the item, while the abscissa axis groups all the items. It illustrates the long-tail phenomenon in recommender systems. The first vertical line separates the 20% most popular items (short-head) from the 80% least popular (long-tail). We observe that the short-head items cumulatively receive many more clicks than the 80% long-tail items. The second vertical line divides the long-tail of the distribution into two parts: the first part we call the long-tail, the second part we call the distant-tail. For cold-start items in the distant-tail, metadata-based recommendation techniques are used.

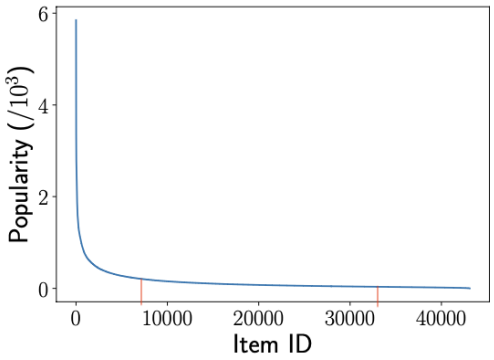


Figure 3: Typical popularity distribution of items depicting the long tail

Popularity bias is a well-known phenomenon in recommender systems where popular items are recommended even more frequently than their popularity would warrant, amplifying long-tail effects already present in many recommendation domains [1]. In other words, most recommendation algorithms favor a few popular items while not giving deserved attention to the majority of others.

2 Softmax Cross-Entropy Loss Regularization

In this section, we present the loss function that we have designed in order to mitigate the popularity bias: the sampled softmax loss regularized by popularity. We start by briefly introducing the accuracy and popularity challenges of the optimization process. We then define the softmax loss function that addresses the first challenge, and we explain how we regularize it with a popularity term to address the second challenge.

2.1 Accuracy and Novelty Objectives

As explained in Section 1, the ultimate goal of the V2V recommendation module is, given a session s_i of videos watched by a user, to provide him with an ordered list of videos he is likely to watch next.

As a reminder, for each input instance, the recommendation module produces a vector $\hat{y}_{raw}$ of dimension equal to that of the catalog of recommendable videos and whose elements are the relevance scores associated with each of the videos in this catalog (not considering the sampling of candidates that will be covered in Section 3). The relevance $relevance(v_i, s_j)$ of recommending a video v_i for a session s_j at timestamp t is given as the similarity score between the predicted next video embedding $\hat{e}(v_{j,t+1})$ and the embedding vector of the given video $e(v_i, s_j)$. The similarity function involved can assume different forms. In this part we restrict ourselves to the standard scalar product and cosine similarity as in equation (8):

$$Relevance(v_i, s_j) = sim(\hat{e}(v_{j,t+1}), e(v_i, s_j)) = \frac{\hat{e}(v_{j,t+1}) \cdot e(v_i, s_j)}{\|\hat{e}(v_{j,t+1})\| \cdot \|e(v_i, s_j)\|} \tag{8}$$

The objective of the learning task is therefore to maximize the pairwise similarity between the predicted next video embedding $\hat{e}(v_{j,t+1})$ and the embedding vector of the next video actually watched by the user in session s_j, denoted $e(v_i^+, s_j)$ (the positive sample), while minimizing the pairwise similarity between the predicted next video embedding and the embedding vectors $e(v_i^-, s_j)$ of the videos that were not watched by the user in this session (the negative samples).

Alongside this accuracy requirement, we also strive to achieve a popularity standard: we want to encourage the recommendation of niche videos by penalizing the incorrect recommendations of popular videos more heavily than the incorrect recommendations of niche videos.

We will now elaborate on these two parts of the optimization step.

2.2 Accuracy Objective: Sampled Softmax Loss Function

Before delving deeper into the popularity regularized sampled softmax loss, let us take a few steps back to define the standard softmax loss. To do this, let us briefly first define the softmax activation function and then the cross-entropy loss function.

The softmax function is frequently used as an activation function on the last layer of neural networks in order to convert the output vector of numbers into an output vector of probabilities ranging from zero to one and whose sum is equal to one. Specifically, when a network produces c values, one for each class of the classification task, the softmax activation function normalizes these c values by converting them into probabilities of belonging to each class.

$$softmax(z)_k = \frac{exp(\gamma.z_k)}{\sum_{i=1}^{c} exp(\gamma.z_i)} \tag{9}$$

Cross-entropy loss (or log loss) on the other hand, measures the performance of a classification model whose output is a probability (typically coming from a sigmoid activation in the case of binary classifications or from a softmax activation in the case of multiclass classifications). It increases as the predicted probability deviates from the actual label. For example, predicting a probability close to 0 when the true label is 1 would lead to a high loss value and therefore a high penalty, while predicting a probability close to 1 would lead to a low loss value and therefore a low penalty. The graph below shows the range of possible cross-entropy loss values given a true observation (y = 0 vs y = 1).

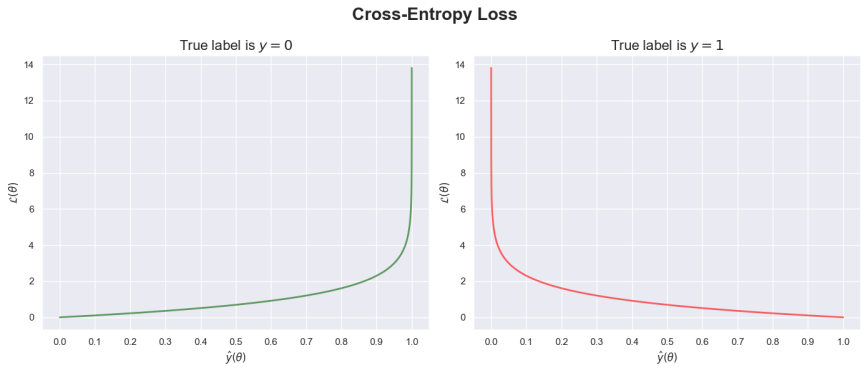


Figure 4: Cross-Entropy loss

In multi-class classification problems, the cross-entropy loss is calculated by considering a separate loss for each class label and then summing the results for each observation.

$$\mathcal{L}(\theta) = -\sum_{i=1}^{c}(y_i \cdot log(\hat{y}_i(\theta))) \text{ where y} \in \{0,1\}^{\text{c}} \text{ and } \hat{\text{y}}(\theta) \in [0,1]^{\text{c}} \tag{10}$$

In the case of binary classification, this is equivalent to applying:

$$\mathcal{L}(\theta) = -(y \cdot log(\hat{y}(\theta)) + (1-y) \cdot log(1-\hat{y}(\theta))) \text{ where y} \in \{0,1\} \text{ and } \hat{\text{y}}(\theta) \in [0,1] \tag{11}$$

Finally, the two components we have just defined, softmax activation on the one hand and cross-entropy loss on the other, are commonly used in sequence to give the softmax loss function.

This is the loss we use to meet our accuracy requirements. We compute the posterior probability $\mathbb{P}(v_i^+$ of a video being the next one given an active user session s with a softmax function over the relevance scores and feed it to a cross-entropy loss. Therefore, the loss takes the form of equation 12 where C is the set of user clicks available for training, whose elements are triples of the form $(s, v_i^+, \mathrm{D_{rv}})$.

$$\mathcal{L}_{accuracy}(\theta) = -\frac{1}{C} \cdot \sum_{(s,v_i^+,\mathrm{D_{rv}})\in C} log(\mathbb{P}(v_i^+|s, \mathrm{D_{rv}})) \tag{12}$$

Since the accuracy loss is differentiable with respect to the θ parameters of the model, we can use backpropagation on gradient-based numerical optimization algorithms in the V2V module. Using this definition, the parameters of the θ model in the V2V module are estimated to maximize the accuracy of the recommendations, which is the first objective.

2.3 Novelty Objective: Sampled Popularity Loss

In order to address our second objective, we propose to add a regularization term to our objective function, which we call popularity loss and whose goal is to control the popularity bias in our recommendations. Note that this term is largely inspired by the work of Gabriel Moreira [12].

In our approach, we consider the definition of novelty [16] defined in equation 13 and based on the logarithmic transformation of popularity:

$$novelty = -log_2(1 + popularity) \tag{13}$$

In order to raise both niche and relevant items in the recommendation lists, we seek to penalize incorrect recommendations according to their popularity: the more popular the recommendation, the more penalized it is. To this end, the novelty of a negative item $novelty(v_i^-)$ is weighted by its probability of being the next item in the sequence $P(v_i^-|s, D_{rv}^-)$. Summing these scores for all negative samples and normalizing the result by the total probability of the negative samples gives the loss for a single instance. It remains to evaluate and sum the losses for each of the input instances to obtain the popularity loss. Note that only the sampled negative items are penalized according to their popularity, the positive items do not affect the evaluation of the popularity loss. Mathematically, the popularity loss term is defined as follows:

$$\mathcal{L}_{novelty}(\theta) = -\frac{1}{C} \cdot \sum_{(s,v_i^+,\mathrm{D_{rv}^-})\in C} \frac{\sum_{v_i\in \mathrm{D_{rv}^-}} \mathbb{P}(v_i|s, \mathrm{D_{rv}^-}) \cdot novelty(v_i)}{\sum_{v_i\in \mathrm{D_{rv}^-}} \mathbb{P}(v_i|s, \mathrm{D_{rv}^-})} \tag{14}$$

where C is the set of click events recorded and used in training and where D_{rv}^{-} contains only the negative elements of our random data samples.

When the videos predicted by the network are popular, the novelty term decreases and tends to minus one, thereby the loss function increases. Conversely, when the predicted videos are niche, the novelty term increases and tends to zero, the loss function decreases.

2.4 Complete Loss Function: Regularized Popularity Sampled Softmax Loss

Finally, the complete loss function we propose to use combines the precision and novelty objectives, as shown in the equation below where λ is the regularization hyperparameter that controls the trade-off between the two fundamental objectives.

$$\mathcal{L}(\theta) = \mathcal{L}_{accuracy}(\theta) + \lambda \cdot \mathcal{L}_{novelty}(\theta); \text{ where :} \tag{15}$$

$$\mathcal{L}_{accuracy}(\theta) = -\frac{1}{C} \cdot \sum_{(s,v_i^+,\mathrm{D_{rv}})\in C} log(\mathbb{P}(v_i^+|s,\mathrm{D_{rv}})),$$

$$\mathcal{L}_{novelty}(\theta) = -\frac{1}{C} \cdot \sum_{(s,v_i^+,\mathrm{D_{rv}^-})\in C} \frac{\sum_{v_i\in \mathrm{D_{rv}^-}} \mathbb{P}(v_i|s,\mathrm{D_{rv}^-}) \cdot novelty(v_i)}{\sum_{v_i\in \mathrm{D_{rv}^-}} \mathbb{P}(v_i|s,\mathrm{D_{rv}^-})}.$$

After developing a loss function that combines the goals of accuracy and popularity, we still need to adjust the λ parameter to achieve the desired effects and therefore balance the accuracy and novelty of recommendations. We describe this work in the next section.

2.5 Effect of the novelty hyperparameter on the loss

Setting the regularization parameter λ to a very high value in our regularized loss strongly penalizes popular recommendations in favor of niche recommendations. Ultimately, we may end up with only niche contents and therefore lose significantly in terms of mean average precision (MAP@k) and recall (Rec@k). Therefore, for the regularization to work properly, we must take care to choose an appropriate lambda regularization parameter. For this reason, we now propose to investigate the effect of the lambda parameter on the performance of the model, both in terms of precision and popularity. We start with a quantitative analysis of evaluation metrics which we then back up with a qualitative analysis.

2.5.1 Quantitative comparison

We aim for a regularization parameter λ for which the novelty improvement is as large as possible while keeping the performance close to constant. To this end, we compare performance for regularization parameters ranging from 0 to 10^8 (the exact values are { 0.1, 1, 10,

100, 200, 250, 400, 500, 600, 750, 800, 1000, 2000, 6000, 8000, 10000, 10^5, $2.5 \cdot 10^5$, $3.5 \cdot 10^5$, $5 \cdot 10^5$, $6.5 \cdot 10^5$, $7.5 \cdot 10^5$, 10^6, $5 \cdot 10^6$, 10^7, 10^8 }). The results we obtain are presented below:

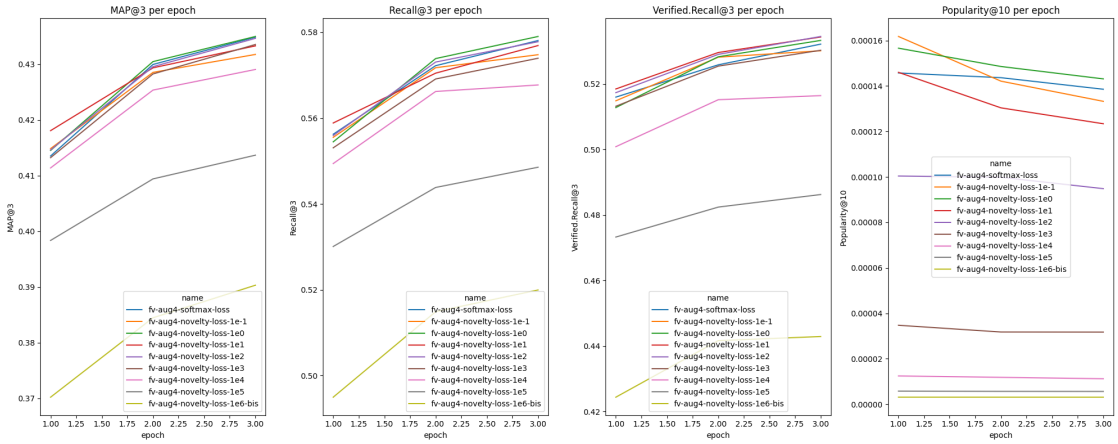


Figure 5: Comparison of baseline and regularized loss performance

For low values ($\lambda \in [0, 1000]$), the superiority of one model over another is not obvious. After repeating the same experiments for these values we found that the gaps in performance were mostly related to the stochastic aspect of the training. However, for larger values, a clear pattern emerges, the evaluation metrics decrease as λ increases. The model then loses precision but gains novelty.

In order to better understand this situation, we propose to study the evolution of the performances as a function of the λ parameter and to add new popularity metrics.

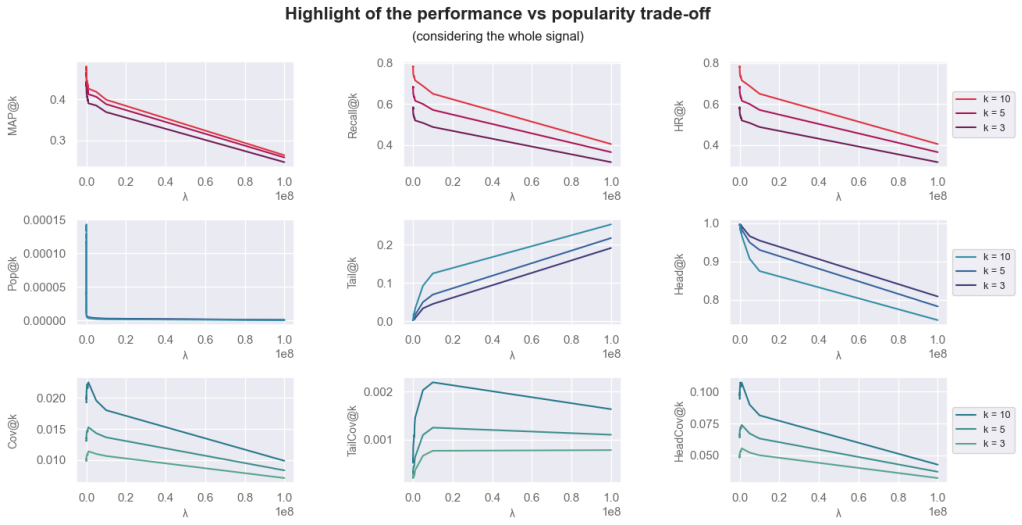


Figure 6: Highlight of the trade-off between precision and novelty ($\lambda \in [0, 10^8]$)

When the regularization parameter approaches one million ($\lambda \sim 1e6$), the accuracy and popularity terms of the loss are of the same order of magnitude ($\mathcal{L}_{accuracy}(\theta) \sim \mathcal{L}_{novelty}(\theta)$) and the optimization stage focuses as much on adjusting the network weights in order to produce an embedding geometrically close to the target embedding than to produce an embedding far from the embeddings of popular items (considering that we use the scalar product as a relevance function). Our primary goal is to make the most accurate predictions and therefore to produce embeddings as close as possible to the target embeddings, the performance of the model must be kept constant or decrease very little. For these reasons the regularization parameter must be less than the million ($\lambda \in [0, 10^6]$).

Figure 7: Highlight of the trade-off between precision and novelty ($\lambda \in [0, 10^6]$)

Regarding popularity, the metrics demonstrate that the effects of regularization are positive on the segment $[0, 10^6]$: not only does the average popularity (pop@k) of the recommended items drop very sharply, but the coverage of items (cov@k, tailcov@k and headcov@k) as well as the proportion of niche videos per recommendation list (tail@k) increase. The hypothesis of a faulty network massively predicting the same subgroup of niche videos or recommending niche videos to the same subset of users is therefore discarded. Beyond these observations, we also observe an important drop in average popularity at k for $\lambda \in [0, 10^5]$ then a stabilization for $\lambda \in [10^5, 10^6]$.

In terms of performance, however, the metrics drop drastically. On the whole segment, $\lambda \in [0, 10^6]$, the maximum difference in MAP@3 is $\delta_{MAP@3,max} = 5.171 \cdot 10^{-2}$. Therefore we have the proportion p of additional correct predictions needed to compensate for $\delta_{MAP@3,max}$ such that:

$$p = 100 \cdot (\frac{\delta_{MAP@3,max}}{AP@3}) \text{ where: } \text{AP@3} = \frac{\sum_{i=1}^{3} P@i \cdot rel_i}{\sum_{i=1}^{3} rel_i} \in \{0.3\bar{3},\ 0.5,\ 1.0\}$$
$$\implies p \in \{5.171\%,\ 10.34\%,\ 15.51\%\}\ .$$

It corresponds to one of the following:

- 5.171% of additional videos correctly predicted and ranked 1,
- 10.34% more videos correctly predicted and ranked 2,
- 15.51% of additional videos correctly predicted and ranked 3.

On the second segment, $\lambda \in [0, 10^5]$, the maximum difference in MAP@3 is this time equal to $\delta_{MAP@3} = 2.837 \cdot 10^{-2}$. This corresponds to one of the following:

- 2.837% (2.122%) of additional videos correctly predicted and ranked 1,
- 5.675% (4.243%) more videos correctly predicted and ranked 2,
- 8.512% (6.365%) of additional videos correctly predicted and ranked 3.

This drop in performance induces prediction errors that are far too important and it is necessary to restrict oneself to lower values of λ and in this case lower than 10^5.

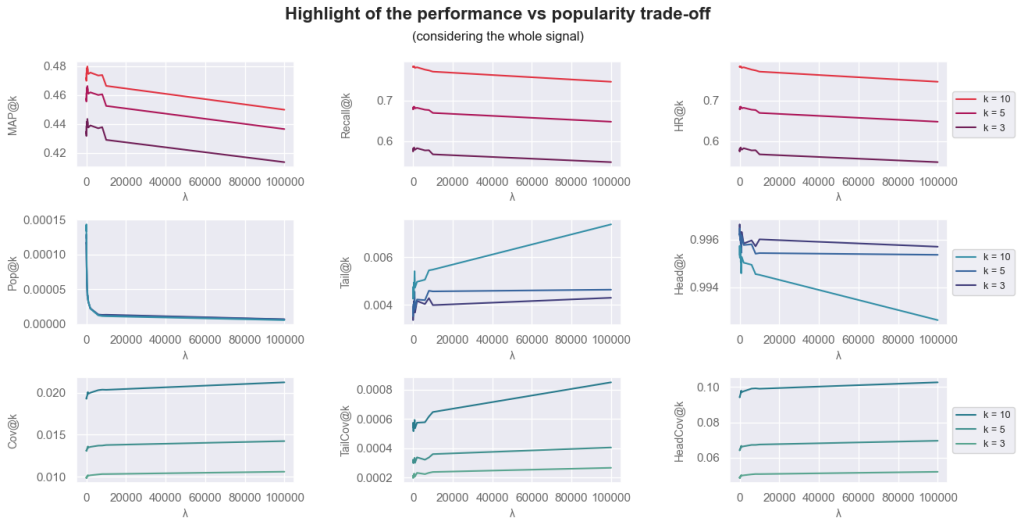


Figure 8: Highlight of the trade-off between precision and novelty ($\lambda \in [0, 10^5]$)

Figure 9: Highlight of the trade-off between precision and novelty ($\lambda \in [0, 10^4]$)

Segment $[0, 10^4]$ (and in particular $[0, 8000]$) seems to be a promising region to find our candidate parameter. On this segment the maximum gap in MAP@3 is equal to $\delta_{MAP@3} = 5.171 \cdot 10^{-2}$, which corresponds to:

- 1.448% (0.5837%) more videos correctly predicted and ranked 1,
- 2.896% (1.167%) of additional videos correctly predicted and ranked 2,
- 4.344% (1.751%) of additional videos correctly predicted and ranked 3.

2.5.2 Qualitative comparison

Until now, we conducted a quantitative survey where we compared and analyzed the results obtained for different regularization parameters. We propose here a qualitative analysis of our results in order to observe more concretely the outcome and to ensure the relevance of the recommendations for the parameters in the selected segment.

The following results prove the effectiveness of regularization. Not only did we observe that the results show that regularization decreases the popularity bias, but this second analysis also confirms that the recommendations remain relevant and linked to the video viewed by the user.

Entrée [8]:

```
candidate_video = CandidateVideo(
    number_candidates = 5,
    predictions_df = all_preds_dfs['predictions_df_fv_softmax_loss_0825'],
    source_blob_name = "video_to_video/run_fv-softmax-loss-0825/vocab/popularity.csv")

candidate_video.call()
```

Figure 10: Generating popular and niche video candidates

■ Retrieving predictions:
  ▶ For experiment fv-softmax-loss-0825
  ▶ For experiment fv-nov-loss-0825-1e8

■ Generating a target video ID:

Candidate input videos:

ID: 1gb8safc67f767dqe1m

Agathe Auproux et Kelly Vedovelli font une danse très sexy dans TPMP et finissent par s'embrasser !

1gb8safc67f767dqe1m

ID: 1gb6r0t4h6gqjt0h6fa

Kelly Vedovelli a été révélée par Hervé Mathoux dans le public du Canal Football Club

1gb6r0t4h6gqjt0h6fa

ID: 1gb8p6kcd546v8240ap

Kelly Vedovelli en vacances : elle s'offre un repos bien mérité après la fin de TPMP (PHOTOS)

1gb8p6kcd546v8240ap

ID: 1gb80sbu14llkdur2pj

Cyril Hanouna se paye Kelly Vedovelli pour son retard dans "TPMP"

1gb80sbu14llkdur2pj

  ▶ Candidates: ['1gb8safc67f767dqe1m', '1gb6r0t4h6gqjt0h6fa', '1gb8p6kcd546v8240ap', '1gb80sbu14llkdur2pj']

  ▶ Choose one of the following target view IDs:1gb6r0t4h6gqjt0h6fa

■ Displaying predictions:

Recommendations

▶ fv-softmax-loss-0825:

Kelly Vedovelli a été révélée par Hervé Mathoux dans le public du Canal Football Club

La blonde qui a fait le buzz derrière Hervé Mathoux était une hôtesse

TROIS MILLE ANS À T'ATTENDRE - Bande-annonce VOST

La danse entre Kelly Vedovelli et Cyril Hanouna !

Retour sur le phénomène "Popstars" avec Bruno Vandelli et Mia Frye

▶ fv-nov-loss-0825-1e8:

Elodie Frégé et Epsilon DON #fetelamour avec AIDES

Plongez dans le trou bleu le plus profond du monde

Surpopulation carcérale : "J'en suis à souhaiter que la CEDH condamne la France"

GILLIAN ANDERSON as DAVID BOWIE in AMERICAN GODS season 1

Zoku owarimonogatari - Bande Annonce □□□□

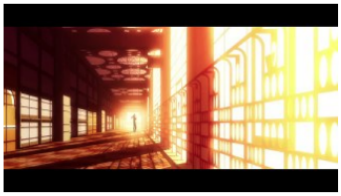


Figure 11: Qualitative analysis first results

Entrée [42]:

```
Generating comparisons with a popular title's substring:

ual_comp = QualitativeComparison(
   run_tokens = ['fv-31-nl-2e3-tf', 'fv-31-nl-1e4-tf', 'fv-31-nl-1e6-tf', 'fv-31-nl-1e8-tf'],
   epoch_idx = 2,
   candidate_substring ='Kelly Vedovelli',
   candidate_exact_match = False,
   get_info = 0,
   up_to = 5)

ual_comp.call()
```

■ Retrieving predictions:
  ▶ For experiment fv-31-nl-2e3-tf
  ▶ For experiment fv-31-nl-1e4-tf
  ▶ For experiment fv-31-nl-1e6-tf
  ▶ For experiment fv-31-nl-1e8-tf

■ Generating a target video ID:

Candidate input videos:

▶ Candidates: ['1gbnv3ook71o1buokq9', '1gbnvvuk34cntvqs8g3', '1gbn3gjhb3qug2lqb20', '1gbn3lamilbcj8q23fg']

▶ Choose one of the following target view IDs:1gbnv3ook71o1buokq9

■ Displaying predictions:

Recommendations

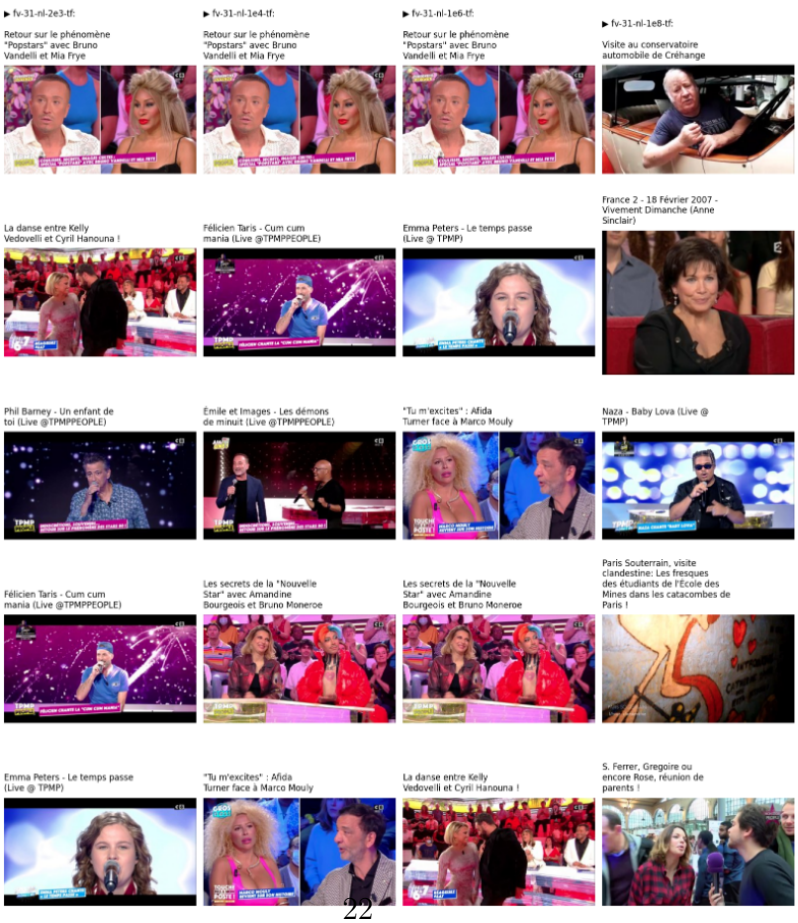


Figure 12: Qualitative analysis second results

3 Candidate Sampling

In general, the field of recommendation faces a major problem: the catalogs of selectable items often contain hundreds of thousands or even millions of items. For its part, Dailymotion hosts hundreds of millions of videos that are viewed by millions of users every day. Given the size of this catalog, it is crucial to be able to select a few samples from the entire catalog for the computation. In this section, we first define the candidate sampling method we use and then study the impact of the number and distribution of candidate samples on precision and recall on the one hand, and on popularity on the other.

3.1 Sampled Softmax Loss

The objective function we have chosen to use includes a sum over all $|V|$ items of our catalog. Not only, for each of the m entries of the training set, the relevance score must be evaluated for the $|V|$ items of the catalog but the normalization of these scores also requires to evaluate the exponential components of these $|V|$ coefficients. This last sum is computationally huge and its evaluation would take $\mathcal{O}(|V|)$ time that would be counted in millions which would be unusable in practice. Therefore, we approximate this sum by sampling the output and therefore computing the score only for a small subset of elements.

Instead of going through the whole catalog for each training step, we sample several negative examples from a noise distribution ($P_n(v)$) whose probabilities correspond to the frequency order of the catalog. The resulting sample is $\widetilde{D}_{rv}$ which includes the singleton formed by the positive element $\{v_i^+\}$ and the proper subset of the selected negative elements $\widetilde{D}_{rv}^-$ such that $\widetilde{D}_{rv} = \{v_i^+\} \cup \widetilde{D}_{rv}^- \subset D_{rv}$. Furthermore, instead of generating separate negative samples for each training instance, we use the same negative samples for an entire batch. The advantage of this approach is that we can further reduce computational time by skipping the sampling. In addition, it also has advantages from an implementation perspective, making the code less complex and speeding up the matrix operations.

In order to include negative sampling into our problem formulation, the loss function becomes:

$$\mathcal{L}(\theta) = -\frac{1}{C} \cdot \left[\sum_{(s,v_i^+,\widetilde{\mathrm{D}}_{\mathrm{rv}}) \in C} log(\mathbb{P}(v_i^+|s,\widetilde{\mathrm{D}}_{\mathrm{rv}})) + \lambda \cdot \sum_{(s,v_i^+,\widetilde{\mathrm{D}}_{\mathrm{rv}}^-) \in C} \frac{\sum_{v_i \in \widetilde{\mathrm{D}}_{\mathrm{rv}}^-} \mathbb{P}(v_i|s,\widetilde{\mathrm{D}}_{\mathrm{rv}}^-) \cdot novelty(v_i)}{\sum_{v_i \in \widetilde{\mathrm{D}}_{\mathrm{rv}}^-} \mathbb{P}(v_i|s,\widetilde{\mathrm{D}}_{\mathrm{rv}}^-)} \right].$$

In this formulation, the choice of the distribution ($P_n(v)$) becomes critical. A lot of studies focus on what makes the best approximation. In practice, what seems to work best is the unigram model raised to the 3/4 power. This parameter is a smoothing parameter whose purpose is to control the degree of "flattening" of the sampling distribution. Set to one it keeps the original distribution, set to zero it distorts it into a uniform distribution. The intuition is that applying such a distortion parameter (3/4) increases the probability of sampling very low probability items but decreases the probability of sampling the most probable items.

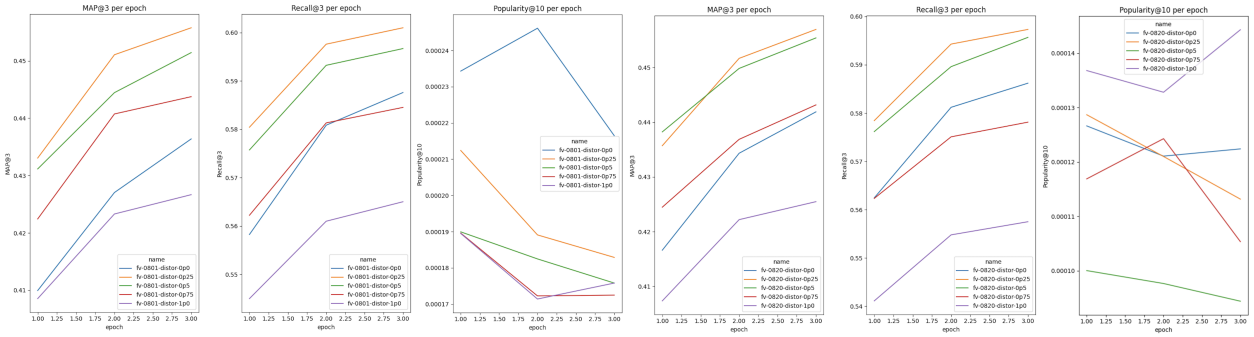
Thus the least popular items are more likely to appear in our sample. In what follows, we motivate our choice of the distortion parameter.

3.2 Estimating the distortion parameter

To improve the quality of the training, the selection of negative samples must be done carefully. Sampling negative items based on their popularity (the number of times they occur in the dataset) generally promotes efficient learning. This is called popularity-based sampling. During the optimization phase, the predicted embeddings are scored according to a given relevance function (the dot product or cosine similarity in our case) with the candidate embeddings, i.e. a target embedding (positive) and n sample embeddings (negative). These scores define the ordering of the videos in the recommendation list. The objective of the optimization is therefore to adjust the weights of the network so that the scores associated with the negative candidates are lower than the scores associated with the targets.

When the selected negative samples are exclusively drawn from the long-tail, the predicted embedding vectors are not confronted with the popular embeddings that are likely to obtain high similarity scores and thus the probabilities associated with the true classes increase, resulting in low penalty and low learning. In other words, the network no longer learns to correctly distinguish true emebddings from other similar emebddings. On the contrary, if negative samples are drawn in the short-head, the predicted embedding vectors are confronted with popular embeddings and thus the probabilities associated with the true classes decrease, resulting in a stronger penalty and learning. The network then correctly learns the ordering of the items in the recommended list. Popularity-based negative sampling thus guides the optimization process to a better generalization state.

Some studies suggest that the use of cross-entropy loss favors either a high distortion parameter with a low number of negative examples or a low distortion parameter with a high number of negative examples. In the experiments below we compare the impact of a distortion parameter chosen in the set $\{0, 0.25, 0.5, 0.75, 1.0\}$. One of the experiments was performed on data from July 16 to August 1, the second one takes data from August 4 to 20.



((a)) Data from 22.08.01 ((b)) Data from 22.08.20

Figure 13: Comparison of performance for different distortion parameters

Following these experiments, we observe that a distortion parameter close to 0.25 seem to lead to better performances on both training sets. We also notice that the superiority of one experiment over the other is not so clear regarding popularity metrics. The ranking of the experiments is not preserved and the mean popularity has decreased by half at the end of August. This is however consistent, and is due to the release in production of a modification in our model at the beginning of August. This led to a decrease in the average popularity of our recommendations and as a result, the data on which our models were trained at the end of August were less popular overall and therefore the average popularity of the recommendation lists in our evaluation set decreased. Finally we chose to continue with a distortion parameter of 0.25 since it seemed to induce better performances.

4 Preference Mechanism

In this section, we are interested in the development of a neural architecture that can determine the preferences of users in order to recommend videos accordingly. We start by briefly presenting the objective and principle of our architecture. We then introduce two settings, one studies the user preferences in a discrete manner, the other studies it in a continuous manner. Finally we analyze and compare the results obtained for each setting.

4.1 A new architecture to determine user preferences

The new architecture is composed of two modules, the recommendation module presented in Section 1 and a new preference module.

As previously, a first layer encodes the input session s_i into an embedded session $e(s_i)$ that is fed to a recurrent neural network. The output h_t of the last GRU cell in the network is concatenated to the embedding $e(context_i)$ which encodes the contextual variables of the session. The resulting vector is then activated twice by a ReLU, first in an element-wise manner and then via a fully connected layer. At this point the V2V module produces, once again, a vector $\hat{y}_{raw}$ which contains the scores associated with the correct class as well as with each of the negative samples.

Instead of applying a softmax function to this vector to obtain a probability distribution used in the optimization process, we would like to adjust it in order to influence the recommendations and to offer content according to the user's preferences between popular and niche. For example, if a user likes niche videos, we will adjust the scores of the $\hat{y}_{raw}$ vector by accentuating those associated with niche videos compared to those associated with popular videos, and vice versa.

In a first stage and in order to simplify the process, we propose a discrete adaptation of our recommendations. In a second stage, we will propose a continuous approach.

4.2 Discrete adjustment of recommendations

In this setup, we divide the set of recommendable videos according to the Pareto principle, into a set of short-head items D_{rv}^H and a second set of long-tail items D_{rv}^T.

In order to determine whether a user prefers niche or popular content, we introduce a preference mechanism alongside our architecture as shown in Figure 16. This mechanism takes the hidden states of each timestamp, which were so far only used in the following hidden states, and feed them to an encoder that we call popularity encoder. In order to embed the nature of items (short-head or long-tail) in their representations, this encoder adds to each hidden state which corresponds to an input item in D_{rv}^T, a vector whose components are equal to a popularity encoding parameter ρ. This parameter is adjustable in order to provide more flexibility to the module.

$$h_i \xrightarrow{\text{Popularity Encoding}} \mathcal{P}_e(h_i) = h_i + \rho.\mathbb{1}_d \iff \Psi(h_i) \in D_{rv}^T \ ,$$

where $\mathbb{1}_d$ denotes a vector of ones and of dimension d, and Ψ denotes a mapping function that maps h_i to the corresponding item v_i.

The popularity encoded hidden states $\mathcal{P}_e(h_t)$ are then fed to an attention mechanism that generates a latent representation of preferences on the current session $e\_session\_prefs$: rather niche or rather popular content. This mechanism uses a Bahdanau attention [4] such as:

$$\alpha_i = W_0 \cdot \tanh(W_1 \cdot \mathcal{P}_e(h_t) + W_2 \cdot \mathcal{P}_e(h_i) + b) \tag{16}$$

$$e_{session\_prefs} = \sum_{i=1}^{t-1} \alpha_i \cdot \mathcal{P}_e(h_i) \tag{17}$$

The preference vector is then concatenated to the last popularity encoded hidden state and compressed into a scalar P via a linear combination of its elements. We expect P to quantify the session preference. A sigmoid function is applied to this preference score P to give coefficient P_{head} which is the correction factor associated with popular videos and from which P_{tail} is deducted.

$$P = [e_{session\_prefs}, \mathcal{P}_e(h_t)] \cdot W_3 \tag{18}$$

$$P_{head} = \frac{1}{1 + e^{-P}} \tag{19}$$

$$P_{tail} = \frac{1}{1 + e^{+P}} \tag{20}$$

The adjustment of the scores of vector $\hat{y}_{raw,i}$ is performed by multiplying the scores either by P_{head} when the score is associated with a short-head video, or by P_{tail} when it is associated with a long-tail video. The adjusted score vector $\mathcal{P}(\hat{y}_{raw,i})$ is finally given to a softmax layer which allows to compute the loss as in the standard architecture.

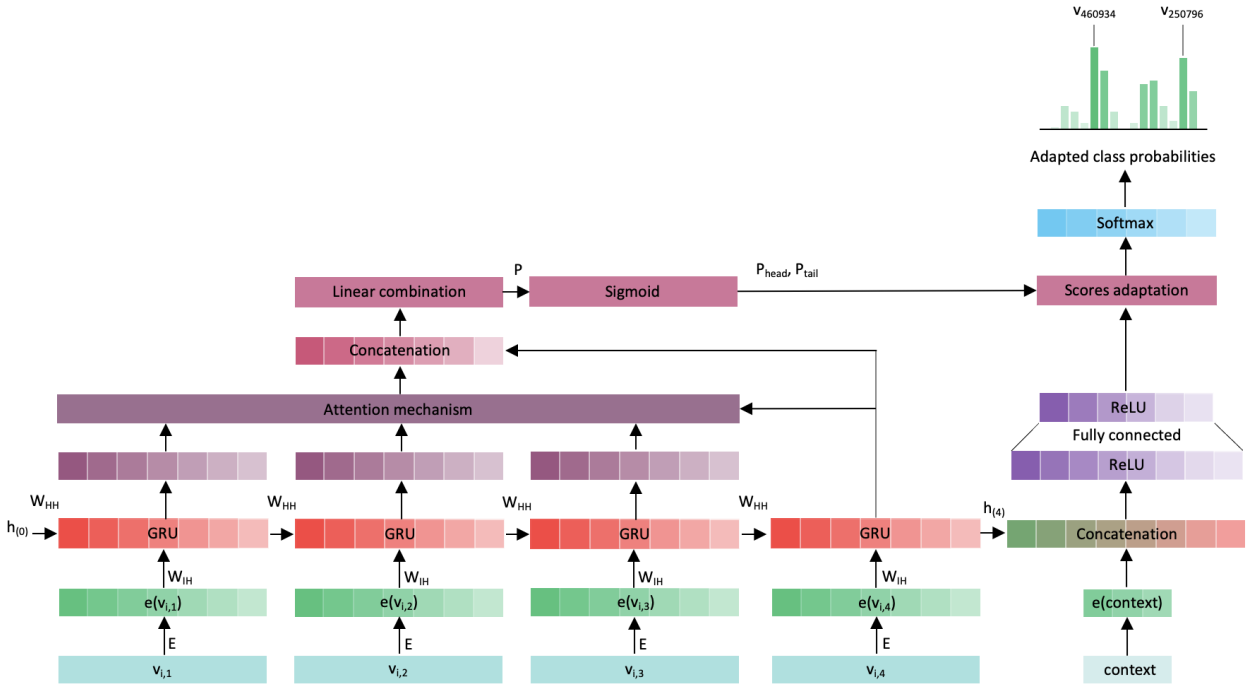


Figure 14: Modified architecture taking the users preferences into account

4.3 Continuous adjustment of recommendations

In this setup, instead of dividing the set of recommendable videos into two sets D_{rv}^{H} and D_{rv}^{T} and determine user preferences for either short-head or long-tail items, we would like to offer a continuous technique.

In order to determine the preference of users, we keep the same preference mechanism alongside our architecture as shown in Figure 16, but replace the popularity encoder. In order to embed the popularity level of items in their representations, the new encoder adds to each hidden state a vector whose components are equal to the popularity of the corresponding input items.

$$h_i \xrightarrow{\text{Popularity Encoding}} \mathcal{P}_e(h_i) = h_i + (1 - pop(v_i)).\mathbb{1}_d \ ,$$

The rest of the preference mechanism remains as before: popularity encoded hidden states are fed to an attention mechanism that generates a latent representation of preferences on the current session. This vector is concatenated to the last popularity encoded hidden state and compressed into a scalar P which quantifies the session preference. Coefficient P_{head} and P_{tail} are deducted through a sigmoid function.

In addition, we also tested a dense layer with up to ten hidden units to obtain ten rectification factors. We also realized a variation in which, instead of using the popularity in the encoding, we used the rank of the element according to its popularity. However, these methods seemed to be working poorly.

4.4 Results:

The different experiments annotated v1 to v6 correspond to different implementations of our mechanism. We focus on version 1 which seems to work as expected. We observe that this architecture significantly decreases popularity and increases coverage by 12%, but in return leads to a loss of 2% to 3% in performance.

Concerning the TailHead@k and Tail@k metrics, we notice that the preference mechanism still allows to exceed the performance of the baseline by 8% and 25% respectively. Moreover we notice that the learning of the preference mechanism is not finished at the third epoch and seems to be in a much better momentum than the baseline on these metrics.

The results are therefore conclusive. A more in-depth study of the threshold considered for the separation of popular or niche videos remains to be done.

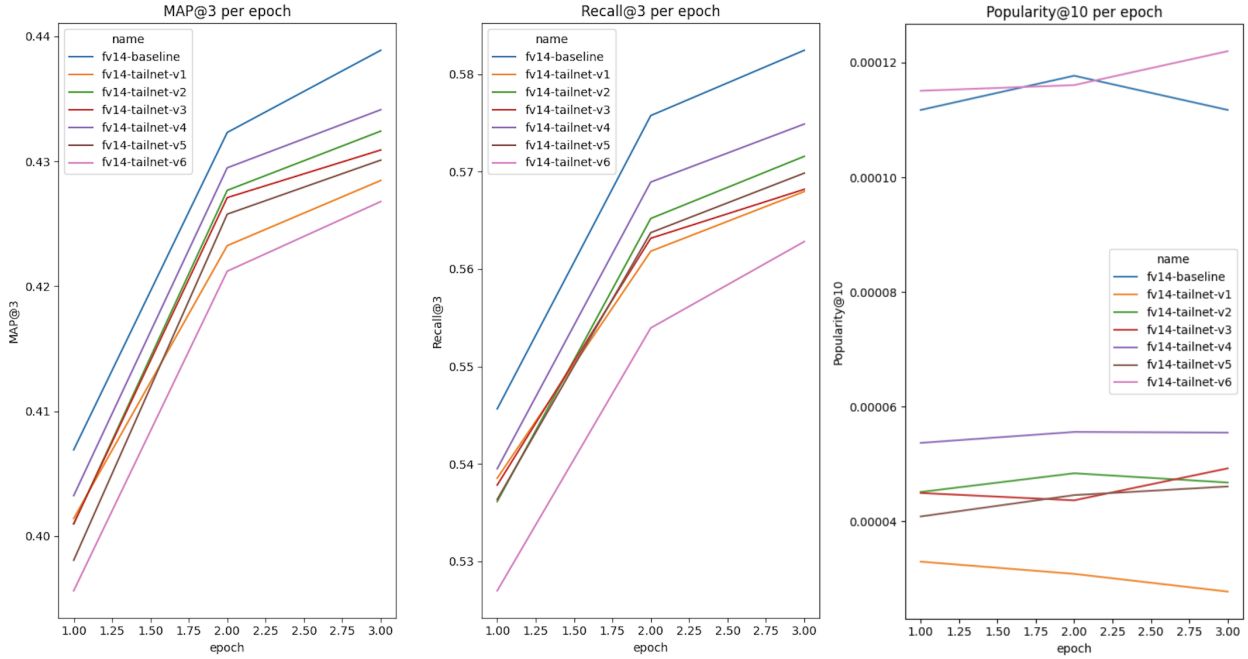


Figure 15: Comparison of the performance obtained with a preference mechanism

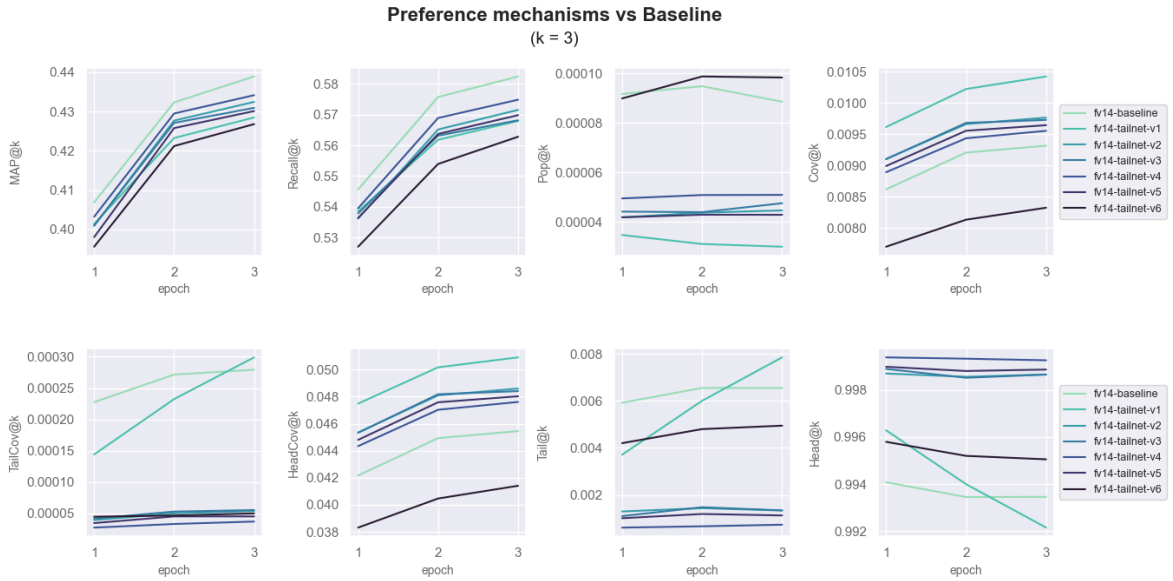


Figure 16: Comparison of the performance obtained with preference mechanisms

5 Scalability

Frequent retraining is often desirable for recommender systems, as new users and content are introduced frequently. Therefore, the model in production must be trainable and capable of producing large-scale results.

Figure 17 provides a comparison of the results from the different experiments. The results are compared to the baseline model in terms of mean average precision, coverage and computation time. Before diving deeper into these comparisons, note that the different studies were conducted at different periods and therefore with different datasets. In the following, we try to draw conclusions from these results while keeping in mind that the execution time and performance still depend on the dataset used.

The baseline is represented in red. For the experiments conducted on both the regularized sampled softmax loss and on the preference mechanism, we average the computation times of the models and select the performances of the best models. Nevertheless, since the dataset used on these specific experiments were the same, the execution times were very close to each other with a standard deviation of $\sigma = 1.24$ for the regularized loss and of $\sigma = ...$ for the preference mechanism. Regarding the experiments conducted on the distortion parameter, we average the results obtained for identical parameters.

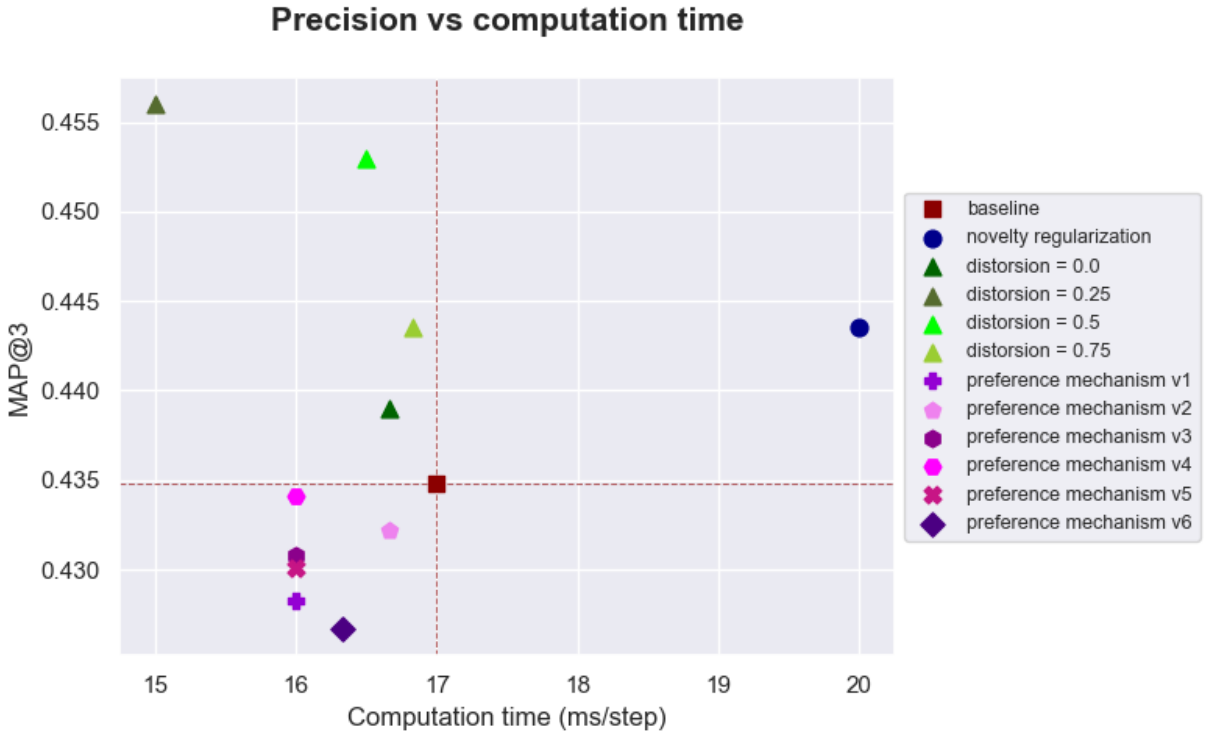


Figure 17: Comparison of models in terms of computing time and performance

In general, the difference in execution time between the variants we had the opportunity to test on an NVIDIA GeForce GTX Titan X GPU is not too high and the training can be done in five to six hours. Only the models based on regularized loss are longer and their training takes almost seven hours.

The approach based on the loss regularization not only allows to significantly recover the catalog but also to gain in mean average precision compared to the baseline. The modification of the regularization parameter λ does not influence the computation time which unfortunately is 17% higher than that of the baseline. Although regularizing the sampled softmax loss increases the learning time, we find that it is not too costly.

The study of the distortion parameters resulted in a substantial improvement compared to the baseline in both evaluation metrics.

Since popularity bias remains a major challenge in recommendation, and since regularized loss outperforms the baseline on all performances except computation time, we suggest using it with an optimal lambda parameter ($\lambda \in [0, 10^4]$).

Computation Time			
Run token	Epoch 1	Epoch 2	Epoch 3
fv-aug4-softmax-loss	6793s (17ms/step)	6767s (17ms/step)	6749s (17ms/step)
fv-aug4-nov-loss-0	7757s (20ms/step)	7596s (19ms/step)	7584s (19ms/step)
fv-0801-distor-0p0	6626s (17ms/step)	6644s (17ms/step)	6712s (17ms/step)
fv-0801-distor-0p25	6042s (15ms/step)	5947s (15ms/step)	5982s (15ms/step)
fv-0801-distor-0p5	6538s (17ms/step)	6563s (17ms/step)	6596s (17ms/step)
fv-0801-distor-0p75	6558s (17ms/step)	6587s (17ms/step)	6624s (17ms/step)
fv-0801-distor-1p0	6845s (18ms/step)	6653s (17ms/step)	6648s (17ms/step)
fv-0820-distor-0p0	6794s (16ms/step)	6740s (16ms/step)	7083s (17ms/step)
fv-0820-distor-0p25	6396s (15ms/step)	6339s (15ms/step)	6351s (15ms/step)
fv-0820-distor-0p5	6888s (16ms/step)	6866s (16ms/step)	6953s (16ms/step)
fv-0820-distor-0p75	7045s (17ms/step)	7048s (17ms/step)	6819s (16ms/step)
fv-0820-distor-1p0	8207s (19ms/step)	8199s (19ms/step)	8193s (19ms/step)
fv-25-tc-nl0tf	8308s (21ms/step)	8212s (21ms/step)	8449s (21ms/step)
fv-25-tc-nl1e10tf	7975s (20ms/step)	7532s (19ms/step)	7462s (19ms/step)
fv-aug4-nov-loss-1000	7935s (20ms/step)	7819s (20ms/step)	7394s (19ms/step)
fv-aug4-nov-loss-200	8352s (21ms/step)	8473s (22ms/step)	8303s (21ms/step)
fv-aug4-nov-loss-400	8501s (22ms/step)	8561s (22ms/step)	8529s (22ms/step)
fv-aug4-nov-loss-4000-b	8618s (22ms/step)	8616s (22ms/step)	8612s (22ms/step)
fv-aug4-nov-loss-600-b	7578s (19ms/step)	7490s (19ms/step)	7650s (20ms/step)
fv-aug4-nov-loss-6000	7950s (20ms/step)	7790s (20ms/step)	7171s (18ms/step)
fv-aug4-nov-loss-800	7765s (20ms/step)	8176s (21ms/step)	7626s (20ms/step)
fv-aug4-nov-loss-8000	7798s (20ms/step)	7869s (20ms/step)	8044s (21ms/step)
fv-aug4-novelty-loss-1e-1	8207s (21ms/step)	8156s (21ms/step)	8214s (21ms/step)
fv-aug4-novelty-loss-1e0	8726s (22ms/step)	8720s (22ms/step)	8648s (22ms/step)
fv-aug4-novelty-loss-1e1	7801s (20ms/step)	7883s (20ms/step)	7890s (20ms/step)
fv-aug4-novelty-loss-1e2	7995s (20ms/step)	7947s (20ms/step)	7900s (20ms/step)
fv-aug4-novelty-loss-1e3	7858s (20ms/step)	7818s (20ms/step)	7818s (20ms/step)
fv-aug4-novelty-loss-1e4	8443s (22ms/step)	8391s (21ms/step)	8423s (21ms/step)
fv-aug4-novelty-loss-1e5	8286s (21ms/step)	8167s (21ms/step)	8055s (21ms/step)
fv-aug4-novelty-loss-1e6-bis	8197s (21ms/step)	8048s (21ms/step)	7847s (20ms/step)
fv-aug4-novelty-loss-1e7	7222s (18ms/step)	7209s (18ms/step)	7194s (18ms/step)
fv-aug4-novelty-loss-1e8	7612s (19ms/step)	7496s (19ms/step)	7444s (19ms/step)
fv-aug4-novelty-loss-2p5e2	7683s (20ms/step)	7075s (18ms/step)	7061s (18ms/step)
fv-aug4-novelty-loss-2p5e5	7510s (19ms/step)	7399s (19ms/step)	7393s (19ms/step)
fv-aug4-novelty-loss-3p5e5	7548s (19ms/step)	7600s (19ms/step)	7604s (19ms/step)
fv-aug4-novelty-loss-5e2	8245s (21ms/step)	8145s (21ms/step)	8313s (21ms/step)
fv-aug4-novelty-loss-5e5	7887s (20ms/step)	7931s (20ms/step)	7870s (20ms/step)
fv-aug4-novelty-loss-5e6	7892s (20ms/step)	7846s (20ms/step)	7834s (20ms/step)
fv-aug4-novelty-loss-6p5e5-bis	7954s (20ms/step)	8147s (21ms/step)	8100s (21ms/step)
fv-aug4-novelty-loss-7p5e2	7956s (20ms/step)	7864s (20ms/step)	8002s (20ms/step)
fv-aug4-novelty-loss-7p5e5	8140s (21ms/step)	8142s (21ms/step)	8085s (21ms/step)
fv-nov-loss-0825-1e8	7822s (20ms/step)	7829s (20ms/step)	7663s (19ms/step)
fv-softmax-loss-0825	6825s (17ms/step)	6690s (17ms/step)	6594s (17ms/step)

Conclusion and Future Work

In this work, we approached the challenge of popularity bias in session-based recommendation from a broader point of view than is usually done. Not only did we perform quantitative studies of our experiments where we analyzed the performance in terms of precision and coverage, but we also performed qualitative studies to concretely visualize the recommendations provided by our recommenders. In addition, we analyzed the scalability of our solutions, which, despite its importance, is often not addressed in the studies we consulted. Our studies were conducted on real-world data that fluctuate from day to day and are subject to the counterfactual feedback loop and therefore influenced by the model in production, namely the baseline. Despite this bias, the best performing methods we have proposed in this work outperform the baseline on all performance metrics except computation time.

We started by documenting the challenges related to recommendation and popularity bias, including user anonymity, data sparisty, cold-start and counterfactual feedback loop. We then proposed a regularization method and analyzed the trade-off between precision and coverage. The proposed loss regularization allowed us to significantly increase the coverage of our catalog (up to 55.65% in Cov@3) but also to slightly increase the precision and recall performances (1.99% in MAP@3 and 1.42% in Recall@3). To further improve the loss, we analyzed the hyperparameters employed when sampling the candidates and concluded that an optimal parameter in our situation was 0.25. Finally, we have proposed a preference mechanism that allows us to determine users' preferences between short-head and long-tail content and to adapt our recommendations accordingly. Our experiments proved the effectiveness of this mechanism in terms of popularity and coverage, but also showed that precision suffered from it.

Our solutions are flexible and can easily be adapted to other session-based recommendation systems. We believe that our results are representative of the effectiveness of our methods and that our work can serve as a basis for other applications of session-based recommendation. However, an important step remains to be done, that of A/B testing to ensure the reliability of our results.

As future work, we would like to further investigate the proposed methods, including an A/B test of the regularized loss and preference mechanism. Depending on the results, we plan to study the pairing of these two methods in a second step. Finally, in the context of regularization we did not question the standard loss used in the baseline. We believe that this loss could be outperformed by the Bayesian Personalized Ranking Loss which compares the score of a positive and a negative item sampled instead of focusing only on the correct class. This work remains to be done.

References

[1] Himan Abdollahpouri. *Popularity Bias in Recommendation: A Multi-stakeholder Perspective*. PhD thesis, 2009.

[2] Himan Abdollahpouri, Robin Burke, and Bamshad Mobasher. Managing popularity bias in recommender systems with personalized re-ranking. 2019.

[3] Gediminas Adomavicius and YoungOk Kwon. Improving aggregate recommendation diversity using ranking-based techniques. In *IEEE Transactions on Knowledge and Data Engineering*, volume 24, 5, page 896–911, 2012.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. 2014.

[5] Alejandro Bellogín, Pablo Castells, and Iván Cantador. Statistical biases in information retrieval metrics for recommender systems. 20:606–634, 2017.

[6] Kyunghyun Cho, Bart Van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. page 1724–1734, 2014.

[7] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In *The International Conference on Learning Representations (ICLR)*, 2016.

[8] Joseph Johnson and Yiu-Kai Ng. Enhancing long tail item recommendations using tripartite graphs and markov process. In *WI Proceedings of the International Conference on Web Intelligence*, page 761–768, 2017.

[9] Komal Kapoor, Vikas Kumar, Loren Terveen, Joseph Konstan, and Paul Schrater. I like to explore sometimes. page 19–26, 2015.

[10] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. 42:30–37, 2009.

[11] Malte Ludewig and Dietmar Jannach. Evaluation of session-based recommendation algorithms. 28:331–390, 2018.

[12] Gabriel Moreira. *Chameleon, a Deep Learning Meta-Architecture for News Recommender Systems*. PhD thesis, 2019.

[13] Yoon-Joo Park and Alexander Tuzhilin. The long tail of recommender systems and how to leverage it. 2008.

[14] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and metrics for cold-start recommendations. page 253–260, 2002.

[15] Lei Shi. Trading-off among accuracy, similarity, diversity, and long-tail: a graph-based recommendation approach. In *RecSys ACM conference on Recommender systems*, page 57–64, 2013.

[16] Saúl Vargas and Pablo Castells. Rank and relevance in novelty and diversity metrics for recommender systems. *Proceedings of the fifth ACM Conference on Recommender Systems (RecSys'11)*, pages 109–116, 2011.

[17] Hongzhi Yin, Bin Cui, Jing Li, Junjie Yao, and Chen. Chen. Challenging the long tail recommendation. In *Proceedings of the VLDB Endowment*, volume 5, page 896–907, 2012.