Real-time Video Recommendation Exploration

Yanxiang Huang[♯,§] Bin Cui[♯] Jie Jiang[§] Kunqiang Hong[§] Wenyu Zhang[§] Yiran Xie[♯]
[♯]Key Lab of High Confidence Software Technologies (MOE), School of EECS, Peking University
[§]Tencent Inc.
[♯]{yx.huang, bin.cui, xie.yiran}@pku.edu.cn
[§]{zeus,kontenhong,gabyzhang}@tencent.com

ABSTRACT

Video recommendation has attracted growing attention in recent years. However, conventional techniques have limitations in real-time processing, accuracy or scalability for the large-scale video data. To address the deficiencies of current recommendation systems, we introduce some new techniques to provide real-time and accurate recommendations to users in the video recommendation system of Tencent Inc. We develop a scalable online collaborative filtering algorithm based upon matrix factorization, with an adjustable updating strategy considering implicit feedback solution of different user actions. To select high-quality candidate videos for real-time top-N recommendation generation, we utilize additional factors like video type and time factor to compute similar videos. In addition, we propose the scalable implementation of our algorithm together with some optimizations to make the recommendations more efficient and accurate, including the demographic filtering and demographic training. To demonstrate the effectiveness and efficiency of our model, we conduct comprehensive experiments by collecting real data from Tencent Video. Furthermore, our video recommendation system is in production to provide recommendation services in Tencent Video, one of the largest video sites in China, and verifies its superiority in performance.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*

General Terms

Algorithms, Performance, Design, Experimentation

Keywords

Real-time, Video Recommendation, Big Data, Online Matrix Factorization, Real-time Collaborative Filtering

SIGMOD'16, June 26–July 1, 2016, San Francisco, USA.

http://dx.doi.org/10.1145/2723372.2742785.

1. INTRODUCTION

The fast development of the Internet has changed everyone's daily life. People spend more and more time on the internet, such as shopping, ticket booking and other entertainment activities. Video sites provide large amounts of videos online, which attract many users with various purposes such as entertainment and news broadcasting. Millions of online videos are played over a billion times every day, by millions of users in YouTube [10], the world's most popular video site. In front of such a huge amount of data, navigating the information in an efficient and satisfying way becomes a hard task for normal users. Personalized recommendation systems are introduced to solve this problem by providing personalized service for every single user, and have been applied in many industrial fields in recent years [35, 34, 8]. As reported by Netflix [9], 75% of the content that people watch follows a recommendation. There is no doubt that video recommendation is of high demand in the era of information explosion. As one of the largest video sites in China, Tencent Video has more than 10 million daily active users and generates about one billion user actions every day. Some unique opportunities as well as challenges are presented for video recommendations in Tencent Video.

Generally, traditional video recommendation systems always produce relevant videos based on video content and user input profiles [33]. However, problems arise when it goes to unregistered users, since there are no profiles for them, while on the other hand unregistered users occupy a large proportion of the total video site users. People made a lot of attempts to improve the recommendation systems. For example, YouTube has introduced its video recommendation system in [10], which is based on user watch activities on the site. Although these works help a lot to improve the quality of recommendation services, they are far from perfect. First, most of the recommendation models are offline and the model training is carried out at regular time intervals. Second, most of the existing real-time recommendation models are too simple to generate satisfactory recommendations. Third, the lack of scalable models applicable to deal with the large amounts of noisy data seriously restrains the application of video recommendation service in the real world. Therefore, it still remains a challenging problem for a more practically efficient real-time video recommendation in the industrial community.

The real world conditions in the Tencent Video introduce multiple challenges for efficient real-time video recommendation. First of all, a large portion of the online users are not registered. It is also important to provide accurate rec-

ommendation services for these users. However, due to the lack of personal information of unregistered users, their profiles are unavailable. We need well-designed models and leverage multiple resources to improve the quality of recommendations. In the literature, collaborative filtering (CF) has achieved great success in recommendation systems [13]. One of CF's most successful techniques is matrix factorization (MF) [19], which assumes user preferences can be modeled by only a small number of latent factors. MF based CF has become more popular since it's robust to the sparsity of the user-video collaborative matrix. However, due to the expensive cost to retrain the matrix factorization model, it is inapplicable to the fast changing trends in the presence of large scale data. Some online learning algorithms address this problem by keeping a representative sample of the data set in a reservoir to retrain the model [13], which however is not appropriate for large streaming data set. To avoid this problem, some other online algorithms propose to update the model based solely on the current observation [31], at the cost of reducing the quality of recommendations. What's more, the most convenient data for MF are high-quality explicit feedback. Unfortunately, the majority of the abundant data in the current online video systems, such as Tencent Video, are implicit feedback, which means that we can only guess users' preferences according to their behaviors. Implicit feedback raises additional challenges for efficient recommendation in Tencent Video. Last but not least, the high computational complexity of recommendation models as well as the massive data in Tencent Video lead to intensive calculations.

In this work, our goal for video recommendation is to provide real-time and accurate recommendation services for users in the condition of large data environment with large amounts of users and videos. Specifically, it demonstrates three requirements. First of all, the recommendation system needs to produce accurate recommendations for users. Second, the model should be updated in real-time to capture users' instant interests in very short delay (in seconds). Third, the processing need to be executed in parallel, i.e., scalable to handle large amounts of computations.

To fulfill the aforementioned goals, we propose a real-time top-N video recommendation system. We develop a real-time MF based CF algorithm, with an online updating strategy. Our online updating strategy takes the confidence levels of different implicit feedback data instances into consideration, where the learning rate in the training process is adjustable to each data instance. To produce top-N recommendations in real-time, we build similar video tables to select high-quality candidate videos for recommendation, which record the most relevant videos for each video. Besides the results of MF, we leverage other factors including video types and time factors to compute the similarity of videos. We present how to implement the model scalably on Storm [30], a distributed real-time computation system for processing streams of data. In addition, some optimization techniques in production are proposed, including demographic filtering and demographic training. We collect real data from Tencent Video to conduct experiments in order to discuss the effectiveness of our model, including the effectiveness of demographic training, and effectiveness of adjustable training. Our video recommendation system is in production at Tencent Video. By providing accurate real-time recommendation services for users, it demonstrates its superiority in performance.

In summary, this paper makes the following contributions:

- We develop a real-time MF based CF algorithm with an adjustable online updating strategy that takes the real-world implicit feedback into consideration.
- To generate top-N recommendations in real-time, we build similar video tables to select high-quality candidate recommended videos, where factors from different aspects are leveraged to compute relevant videos.
- We present some particular issues in production for our model, including scalable implementation, and two optimization techniques including demographic filtering and demographic training.
- We demonstrate the effectiveness of our model by conducting comprehensive experiments on real data and verify its superiority by deploying our recommendation system in production.

The rest of this paper is organized as follows. In Section 2, we review the related work. We present the real-time MF based CF algorithm in Section 3. Section 4 describes the procedure of real-time top-N recommendation generation where different factors are leveraged to compute relevant videos. Deployment issues along with some optimizations are discussed in Section 5. In Section 6, we investigate the effectiveness of our model and demonstrate its superiority in production. Finally, we conclude our work in Section 7.

2. RELATED WORK

There are two research fields closely related to our work, including video recommendation and online matrix factorization. Here we briefly review previous works on the corresponding literatures.

2.1 Video Recommendation

There exist many research works on video recommendation. In 2010, YouTube introduced its video recommendation system in [10], which recommends personalized sets of videos to users based on their activities on the site. In [37], the authors employed social and content information of videos, and proposed methods for video recommendation in sharing community, which fused the content relevance and social relevance to identify the relevant videos. To address the limitations of traditional Latent Factor Models and neighborhood-based collaborative filtering, Cui et al. [9] represented users with content attributes of videos, and represented videos with users' social attributes, so that both users and videos can be represented in a common space concatenated by social attributes and content attributes. In [11], the authors found candidate recommended videos by detecting the hot topics in Twitter, and generated final video recommendations by considering the user profile in YouTube, time factor, and quality factor. VecLp was proposed in [14], which recommends relevant Internet videos to the subscribers by collecting information from the live TV programs. In [23], the authors constructed user profiles as an aggregate of tag clouds and generated video recommendations according to similar view patterns. Mei at el. [33,

21] proposed video recommendation system based on multimodal fusion and relevance feedback. They combined features from three aspects including textual, visual, and aural to represent videos and expressed the relevance between two videos as the combination of relevance from three aspects. Chen et al. [5] provided personalized video suggestions for users on video search engines by exploring a tripartite graph over (user, video, query). They developed an iterative propagation scheme over the tripartite graph to compute the preference information of each user.

2.2 Real-time Collaborative Filtering

Recently, real-time collaborative filtering (CF) has attracted growing attention [27, 13, 36, 6]. Collaborative filtering can be divided into neighborhood-based CF and model-based CF. The neighborhood-based CF includes the user-based CF methods and the item-based CF methods, while the typical model-based CF is Latent Factor Models [19, 18] where matrix factorization (MF) techniques are usually used to solve the problem.

Real-time Neighborhood-based CF: StreamRec was proposed in [3] which implements a scalable item-based CF recommender model based on a stream processing system. In [15], the authors developed incremental and parallel versions of a co-clustering algorithm, and used it to build a real-time collaborative filtering framework, which predicts user-item ratings based on the average rating of co-clusters, users' bias and items' bias. To address the scalability problem of traditional user-based CF, Papagelis et al. [22] proposed a method to incrementally update user-to-user similarities by dividing the similarity computation into parts where each part could be computed incrementally. Similar strategies were used in [26], where the authors described a large scale implementation of a video recommendation system based on item-based CF, making use of a commercial cloud computing platform. A practical item-based CF was proposed in [17], with scalable incremental update mechanism considering several problems in real world production. However, it has been proven that model-based CF could achieve better performance than neighborhood-based CF in most cases [19]. Therefore, we focus on the matrix factorization method, one of CF's most successful techniques.

Real-time Model-based CF: Sarwar et al. [29] proposed an incremental singular value decomposition algorithm that could add new user or item vectors into the latent factor space utilizing the fold-in method. In [1], the authors proposed an online matrix factorization that could incorporate users' or items' properties. Ling et al. [20] developed an online learning framework consisting of probabilistic MF and ranking MF. In [27], an online regularized kernel MF model that can incrementally add new users or new items was proposed for large-scale recommender systems. Diaz et al. believed that recommendations based on pairwise ranking methodology could produce better results [13, 12]. In [12], to solve the short-term "memory" problem in online learning approaches, they proposed using the reservoir to sample the whole history data for training model. And in [13], they further developed a selective sampling strategy to perform online model updates based on active learning principles. Vinagre et al. [31] introduced a simple but fast incremental MF algorithm for positive-only feedback, which retrains the model at each new observation in a single step. In [32], the authors built a dynamic online MF model based on implicit user feedback stream, where whether a user was interested in an item or not was dependent on their adoption tendency.

3. ONLINE ADJUSTABLE MATRIX FACTORIZATION FOR IMPLICIT FEEDBACKS

In this section, we introduce our online adjustable matrix factorization for implicit feedbacks.

3.1 Basic Matrix Factorization

Collaborative filtering (CF) has achieved great success in recommendation systems. As one of CF's most successful techniques, matrix factorization (MF) [19] assumes user preferences can be modeled by only a small number of latent factors and tries to explain the observed user-item ratings by these factors inferred from the rating patterns. MF based CF has become popular because of its good scalability, good predictive accuracy, and robust to the sparsity of the user-video collaborative matrix. In addition, it offers much flexibility for modeling various real-life situations. So far, MF methods have proven to be generally superior to other alternatives.

Matrix Factorization models map both users and items to a joint latent factor space of dimensionality f, so that user-item interactions are modeled as inner products in that space. The goal of MF is to uncover the latent factors that can explain observed ratings. Specifically, each user u is associated with a vector $x_u \in \mathbb{R}^f$, and each item i is associated with a vector $y_i \in \mathbb{R}^f$. For a given item i, the elements of y_i measure the extent to which the item belongs to those factors. Similarly, for a given user u, the elements of x_u measure the extent of interest the user has in items that are high on the corresponding factors. We represent the user u's rating of item i as r_{ui}, which indicates the preference by user u of item i. Higher r_{ui} values mean stronger preference, and the goal of MF is to predict the unknown ratings. The prediction is done by taking an inner product, i.e., $\hat{r}_{ui} = x_u^T y_i$. The major challenge of MF is to compute the mapping of each user and item to factor vectors $x_u, y_i \in \mathbb{R}^f$. To learn the factor vectors (x_u and y_i), the training is performed by minimizing the regularized squared error on the set of known ratings:

$$\min_{x_*, y_*} \sum_{(u,i)\in D} (r_{ui} - x_u^T y_i)^2 + \lambda(\| x_u \|^2 + \| y_i \|^2) \quad (1)$$

where D is the set of user-item pairs for which r_{ui} is known, i.e., the training set, and λ is a parameter used to control the extent of regularization. The regularization term $\lambda(\| x_u \|^2 + \| y_i \|^2)$ is used to avoid overfitting, and the value of λ is data-dependent and usually determined by experiments.

Equation 1 tries to capture the interactions between users and items that produce the different rating values. However, much of the observed variation in rating values is independent of the interactions, yet is affected by the effects associated with either users or items, known as biases or intercepts. For example, some users tend to give higher ratings than others, and some items tend to receive higher ratings than others. Considering the biases effect, we extend the prediction of rating r_{ui} as follows:

$$\hat{r}_{ui} = \mu + b_u + b_i + x_u^T y_i \quad (2)$$

where μ denotes the overall average rating, and the parameters b_u and b_i indicate the observed deviations of user u and

item i respectively, from the average. The observed rating is thus broken down into four components: global average, item bias, user bias, and user-item interaction. Then the objective function of the model for training, i.e., Equation 1 should be extended to:

$$\min_{x_*,y_*,b_*} \sum_{(u,i)\in D} (r_{ui} - \mu - b_u - b_i - x_u^T y_i)^2 + \lambda(\| x_u \|^2 + \| y_i \|^2 + b_u^2 + b_i^2) \quad (3)$$

The most successful methods to solve MF optimization problem are Alternating Least Squares (ALS) [2] and Stochastic Gradient Descent (SGD) [1]. It has been shown [24] that SGD based optimization generally performs better than ALS when using sparse data, both in terms of model accuracy and running time efficiency. Given a training dataset consisting of tuples in the form <user, item, rating>, SGD performs several passes through the dataset until some stopping criteria is met, which is typically a convergence bound or a maximum number of iterations. At each iteration, SGD sweeps over all known ratings in the training set. For each rating, SGD updates the parameters by correcting them in the inverse direction of the gradient of error. The extent of correcting is determined by a factor of $\eta <= 1$, known as step size of the learning rate. Based on Equation 2, the associated prediction error of rating r_{ui} is computed as follows:

$$e_{ui} = r_{ui} - \mu - b_u - b_i - x_u^T y_i \quad (4)$$

The updating operations are performed as follows:

$$\begin{aligned} b_u &\leftarrow b_u + \eta(e_{ui} - \lambda b_u) \\ b_i &\leftarrow b_i + \eta(e_{ui} - \lambda b_i) \\ x_u &\leftarrow x_u + \eta(e_{ui} x_u - \lambda x_u) \\ y_i &\leftarrow y_i + \eta(e_{ui} y_i - \lambda y_i) \end{aligned} \quad (5)$$

3.2 Implicit Feedback Solution

We first introduce our solution to the implicit feedback problem, which to a large extent dictates our design of the online adjustable matrix factorization (MF) algorithm.

The most convenient data for matrix factorization are high-quality explicit feedback, i.e., explicit input by users regarding their interest in products, such as the star ratings, and thumbs-up or thumbs-down comments. However, explicit feedback is not always available in the practical situations. Actually, the majority of the abundant data are implicit feedback, i.e., user behaviors that indirectly reflect user opinions. Since we can only guess users' preferences according to the user behaviors, implicit feedback may reduce the recommender systems' performance if not appropriately handled [16].

There are various types of user behaviors in Tencent Video, including click, watch, comment, etc. The data that we have at our disposal is quite noisy since it measures a user's engagement and happiness indirectly. For example, the fact that a user watched a video in its entirety is not enough to conclude that he actually liked it, while a user may watch a favorite video for just a short period because of time limitation. Both the video length and the user engagement level influence the signal quality. Based on the intuition that different user actions represent different degrees of user's interests, we set proper weights to users' various action types. For example, a click behavior may correspond to a one star rating while a comment behavior equals a three star rating. The weights for various user actions are determined by experiences according to the real world condition in the application. Table 1 shows our settings in Tencent Video taking some action types as example. The $Impress$ action represents video i is displayed to user u. And the $Play$ action refers to that user u begins to watch a video i, while the $PlayTime$ action reports the time period of video i viewed by user u.

Table 1: User Action Weight Settings

Actions	Impress	Click	Play	PlayTime
Weights	0	1	1.5	[1.5,2.5]

Specially, for users' $PlayTime$ actions, we set their weights according to the percentile view time length, as follows:

$$w_{ui} = a + b\log(vrate_{ui}) \quad (a \geqslant b, 1 \geqslant vrate_{ui} \geqslant 0.1) \quad (6)$$

where $vrate_{ui} = t_{ui}/t_i$, t_{ui} is the user's viewing time and t_i denotes the full time length of the video.

We use view rate instead of user absolute viewing time to eliminate the variation on time length of videos of various types. To reduce the noise due to the various messy implicit feedback, we limit $rate_{ui} \geqslant 0.1$. In other words, we treat $PlayTime$ actions with $vrate_{ui} < 0.1$ as inefficient ones that we set their weights the same as $Play$ actions. Since we can hardly infer a user's real preference according to a stop watching action, it is unwise to generate a negative feedback, considering the diversity of recommendation. Actually, we have tested some alternatives such as $w_{ui} = a + b * vrate_{ui}$, and Equation 6 gave the best performance.

To train the MF model, we need r_{ui} values that indicate the preferences of user u for video i. For implicit feedback, a natural way is to treat the weights of user actions as the corresponding ratings, i.e., $r_{ui} = w_{ui}$, which is utilized in previous work [17]. However, although it works well in the item-based CF, there arise serious problems in the MF model, which we will illustrate in our experiments in Section 6.1.2. We attribute the poor performance to the noise raised by the inappropriate guess of implicit feedback, since MF is sensitive to the noisy data. Therefore, we turn to an alternative method. To be specific, we set r_{ui} to be binary variables, and denote the weight of user action w_{ui} as the confidence it measures, as inspired by the work in [16].

$$r_{ui} = \begin{cases} 1 & w_{ui} > 0 \\ 0 & w_{ui} = 0 \end{cases} \quad (7)$$

Simply speaking, a user u's interaction with video i ($w_{ui} > 0$) leads to an indication that u likes i($r_{ui} = 1$). On the other hand, if u never interacts with i, we believe no preference ($r_{ui} = 0$). Furthermore, our beliefs are associated with varying confidence levels, i.e., the user action weights w_{ui}, which will be utilized for the adjustable online updating strategy in the following section.

3.3 Adjustable Online Updating Strategy

As illustrated in Section 3.1, traditional matrix factorization (MF) is trained in a batch mode, which requires several passes or iterations through the dataset to train a model. As timeliness becomes more and more important in this rapidly

[1] http://sifter.org/ simon/journal/20061211.html

changing world, traditional training method is not applicable for streaming data where ratings are continuously generated. Therefore, we propose a real-time MF algorithm with adjustable online updating strategy based on SGD.

First of all, our real-time video recommendation system requires a model that is able to capture users' recent activities and affect the recommendation results in real-time. To handle the huge amounts of streaming data where unbounded number of user actions increases, we need an incremental updating strategy that does not require iterations through the dataset. For the incremental updating strategy, each user action will be processed only once, and no iterations can ensure the model is converged. Actually, as the user actions are continuously generated at an unpredicted rate, the model will be updated all the time, thus is always not converged in terms of streaming data, which arises additional challenges for the real-time model training. Specially, the model in our streaming data condition should satisfy two requirements: able to produce satisfactory recommendations and always sensitive to new user actions. To be specific, first, the model needs to represent the attributes of users and videos to predict accurate preferences. Second, new user actions should have influence on the model in real-time, i.e., the model should quickly adapt to the new user actions, instead of staying the local optimal state of the historical data. These two requirements tend to conflict to some extent. To produce accurate recommendations at present, the model should adapt to current historical data, but this leads to its insensitivity to the new user actions, which may decrease its performance in the future. An efficient real-time MF model must take both aspects into consideration.

In this context, we propose an adjustable incremental SGD algorithm that can update the MF model in real-time to different extents for different user actions. We leverage the confidence levels of user actions in last section and reflect the difference of user actions in the updating strategy for incremental training. On one hand, for user actions with low confidence levels, which are more likely to be noise data, we decrease their influence on the model, to avoid overfitting the noise data or going to the wrong optimizing direction, thus sensitive to new user actions. On the other hand, the influence of user actions with high confidence levels is increased, which are more likely to represent users' true preferences, so that the model can represent attributes of users and videos, thus able to produce good performances. We employ the learning rate of the updating strategy to control the influence on the model, which is proportional to the confidence levels.

To be specific, we update the model at each new user action in a single step, no iterations required. For each user action, its influence on the model is dependent on both its rating and confidence level. We update the model only when the rating $r_{ui} = 1$, i.e., the impression records of users (video i is displayed to user u, $r_{ui} = 0$) do not have influence on the model. For user actions with $r_{ui} = 1$, the higher the confidence w_{ui}, the more influence on the model training procedure, i.e., a larger learning rate. Specifically, we update the model for different user actions according to their confidence levels, as follows:

$$\eta_{ui} = \eta_0 + \alpha w_{ui} \tag{8}$$

where η_0 is the basic learning rate of the training process, and α is a parameter to control the influence of confidence level on learning rate, which is determined by experiments.

The updating algorithm is shown in Algorithm 1. Since we use streaming data set in our scenario and user rating actions are generated at unpredicted rate, we just show the updating strategy for one user rating action for simplicity. For each user rating action, we first compute its preference and confidence, r_{ui} and w_{ui}. Providing the preference is positive ($r_{ui} = 1$), we will update the model according to the confidence level of the user rating action by Equation 8. Every user action will affect the model in real-time, and influence the recommendation results immediately. For new users or new items, we first initialize their vectors before the updating operation. We employ a distributed memory-based key-value storage to store the x_u and y_i vectors, as we will illustrate in Section 5.1. Thus new users and items can be easily added.

Algorithm 1: Real-time Matrix Factorization Model Update based on SGD

Input: user rating action tuple recording user u, video i and action information

Compute r_{ui} and w_{ui}
if $r_{ui} = 1$ **then**
  if *u is new* **then**
    Initialize x_u
  end
  if *i is new* **then**
    Initialize y_i
  end
  Compute η_{ui} by Equation 8
  Compute e_{ui} by Equation 4
  $b_u \leftarrow b_u + \eta_{ui}(e_{ui} - \lambda b_u)$
  $b_i \leftarrow b_i + \eta_{ui}(e_{ui} - \lambda b_i)$
  $x_u \leftarrow x_u + \eta_{ui}(e_{ui}x_u - \lambda x_u)$
  $y_i \leftarrow y_i + \eta_{ui}(e_{ui}y_i - \lambda y_i)$
end

4. REAL-TIME VIDEO RECOMMENDATION GENERATION

In this section, we introduce the procedure from receiving user recommendation request to generating real-time video recommendation in our recommendation system. Moreover, strategies to generate high-quality candidate videos are described.

4.1 Recommendation Request Processing

The goal of our recommendation system is to provide personalized recommendations that help users find high-quality videos consistent with their interests. In order to improve the degree of users' satisfaction, it is imperative that these recommendations are provided in real-time and reflect users' recent activities on the site. This requirement prevents us from generating recommendations by pre-computation that is adopted in many works, because pre-computed recommendations cannot reflect users' recent activities. Instead, we compute new recommendations for users in real-time for each user request.

The procedure from receiving user recommendation request to generating real-time video recommendation is shown in Figure 1, where real-time denotes that the delay is generally in mill-seconds. A typical situation where the recommendation happens is that when a user opens a video to watch, we need to produce similar videos as recommendations. After receiving the user recommendation request, we first get several seed videos and then expand the set of their relevant videos to select some candidates. Afterwards, we get the user & video vectors computed by MF model, and predict the preference of the user for these candidate videos according to Equation 2. Final recommendations are generated by ranking these candidate videos according to the predicted preferences. Specifically, example seed videos may be the video user currently watching in the aforementioned situation or historical videos a user interacted with (watched, liked) before in other scenarios where the user is not watching any videos. In addition, a distributed key-value storage is utilized to store the results of the MF model or user historical data, which we will illustrate in detail in Section 5.1.

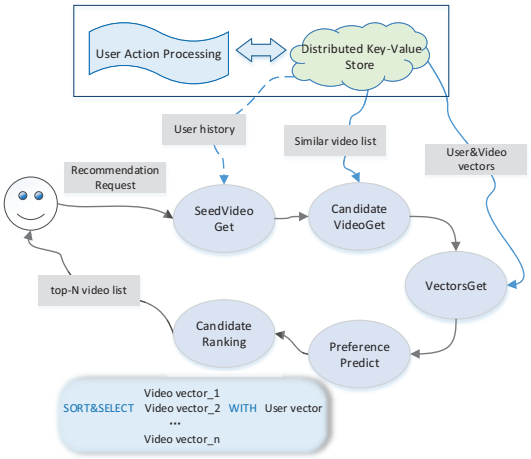


Figure 1: Procedure of Real-time Video Recommendation Generation

Note that we select some candidate videos to generate final recommendations by further ranking. The candidate videos selection is vital in real-time recommendation generation, since the computation through the whole video set is too expensive and time-costly to respond to users in real-time. Considering the dimension of user or video vector always ranges from 20 to 200, the preference prediction that takes the inner product of vectors is definitely a computing intensive task. In a large video site like Tencent Video, the whole video set contains millions of videos, and there are more than 1 billion user requests every day, with maximum 0.1 million requests in one second. The huge computation in real-time video recommendation generation will be a disaster without the candidate videos selection.

4.2 Candidate Video Preparation

From the discussion in above section, we can conclude the key of real-time video recommendation generation is the candidate videos selection. As illustrated previously, candidates are generated by expanding the set of relevant videos of seed videos. Therefore, to select candidate videos of high quality, we need an accurate mapping from a video i to a set of similar or relevant videos. In this context, we define similar videos as those that a user is likely to watch after having watched the given seed video i. To obtain the accurate similar videos, we combine the following three factors, collaborative filtering similarity, type similarity, and time factor.

4.2.1 Collaborative Filtering Similarity

Based on the matrix factorization model computed in Section 3, we can get a vector of latent factors $y_i \in \mathbb{R}^f$ for each video i. Considering the indication of these vectors, we can naturally compute the similarity between video i and j by taking an inner product, as follows:

$$s1_{ij} = y_i^T y_j \tag{9}$$

4.2.2 Type Similarity

In Tencent Video, every video has its type in a fine-grained category system. It is widely shared that same type videos are likely to be more similar than different types. Thus we take videos' types into consideration when computing similar videos. For video i and j, the type similarity is defined as follows:

$$s2_{ij} = \begin{cases} 1 & type(i)\ equals\ type(j) \\ 0 & type(i)\ not\ equals\ type(j) \end{cases} \tag{10}$$

4.2.3 Time Factor

To track the real-time changing trends in video recommendation, we add time factor into videos' similarity computation. We hold the view that the similarity between video pairs is time-sensitive and will descend as time goes on without new relevant user actions support. In other words, the past similar videos should be gradually forgotten. Therefore, we introduce time factor in the computation of similarity.

To be specific, the similarity of video pairs decreases as their update time goes far away from the current time. The update time refers to the time of the latest user action that touches off the similarity computation (similarity of video i and j will be updated only when new user action relating to i or j happens). Formally, we use a damping factor to measure this time factor, as follows:

$$d_{ij} = 2^{-\Delta t/\xi} \tag{11}$$

where Δt is the time difference between sim_{ij}'s(defined in the following section) update time and current time, and ξ is a parameter to control the rate of decay.

4.2.4 Fusion

In our context, each aforementioned similarity factor contributes to the final recommendation to different extent. We perform the relevance fusion by taking the advantage of each similarity factor. Given two videos i and j, the overall relevance between them is defined as:

$$sim_{ij} = d_{ij}((1-\beta) * s1_{ij} + \beta * s2_{ij}) \tag{12}$$

where β is a parameter adjusting the weight of different similarity factors in the final relevance function. The exact values of ξ and β are determined by the experiments.

5. DEPLOYMENT ISSUES

In this section, we present some deployment issues, including how to implement the model scalably to accomplish huge computations evolving large amounts of streaming data, and several optimization techniques in production, including demographic filtering and demographic training.

5.1 Scalable Implementation

We implemented the algorithm based on Storm [30, 7], a real-time stream data processing system. The primary concept in Storm is stream, which represents an unbounded sequence of data tuples. Storm provides "spouts" and "bolts" as the basic operations to conduct stream processing. A spout is responsible for producing the input streams, and passing the data to bolts. A bolt may consume any number of input streams and transform those streams in some way. Bolts can also emit new streams and pass them to some other bolts. Each spout or bolt can be executed in parallel, whose parallelism can be set by configuration. Complex stream processing may require a graph of multiple spouts or bolts where the edges indicate how the streams should be passed around. The graph is called a "topology", which is what we submit to Storm for real-time computation. A topology will process messages forever unless it is killed.

The topology framework of our real-time incremental matrix factorization based collaborative filtering is shown in Figure 2. The parallelism of different spout or bolts is determined by the data set. The grouping mode (denoted by "*value*" after colon in the figure) defines how to distribute the stream data to different workers of a spout or bolt. For example, in Figure 2, the spout emits data tuples in the form of $< user, video, action >$ and the $UserHistory$ bolt receives these data tuples that data tuples with the same user id will go to the same worker to be processed. In addition, we employ a distributed memory-based key-value storage (KVStore in the figure) to store the status data used in computations. We can access a vector x_u or y_i by its corresponding key (the user id or video id), without influencing other vectors. In this case, the KVStore helps distribute the computations among operators.

Specifically, the spout gets data from Tencent Video, parses the raw message, filters the unqualified data tuples, and transforms data tuples to the next bolts. The data tuples spout generated contains several fields, such as user id, video id, action type, action result, etc. For clear presentation, we just show the main fields of data tuples in the figure. The user action tuples are grouping by user id field, and flow to the next bolts. There are three main lines to process the user action tuples in parallel. First, the $ComputeMF$ bolt and $MFStorage$ bolt receive the user action tuples and update the vectors (x_us and y_is) in KVStore according to Algorithm 1. Second, the $UserHistory$ bolt receives the user action tuples and records users' behavior histories in KVStore. Third, the $GetItemPairs$ bolt, $ItemPairSim$ bolt and $ResultStorage$ bolt receive the user action tuples, do the video pair similarity computation utilizing user histories recorded by $UserHistory$ bolt, and finally store a top-N similar video list for each video in KVStore.

Note that the model updating is done by $ComputeMF$ and $MFStorage$ two bolts. First, the user action tuples are transferred to $ComputeMF$ bolt. $ComputeMF$ bolt reads user vectors and video vectors from KVStore and computes new vectors as in Algorithm 1. Then the new vectors will be sent to $MFStorage$ bolt, grouping by their corresponding keys used in KVStore storage. $MFStorage$ bolt receives the new vectors and writes them to KVStore. The key factor in this implementation design is the reassignment of new vectors from $ComputeMF$ bolt to $MFStorage$ bolt, grouping by their keys, which guarantees only a single worker node should operate over a specific video or user vector at some point. Using this method, the updating of vectors is atomic that no write conflict would happen. Thus the calculation can be safely scaled.

Video pair similarity computation is done by three bolts, including the $GetItemPairs$ bolt, $ItemPairSim$ bolt and $ResultStorage$ bolt. Different from model updating, the goal of this design is to reduce resource consumption and speed up computing. To be specific, by transferring the data streams with fields grouping, the queries of the same key for KVStore can be reduced to the same workers and thus we can employ optimizations for efficient processing. Some possible optimizations may be the *combiner* technique and the *cache* technique [17].

5.2 Some Optimizations

In this section, we introduce some useful optimizations in the production.

5.2.1 Demographic Filtering

One of the recommendation metrics is diversity and novelty. YouTube [10] expands the candidate video set by taking a limited transitive closure over the related videos graph to generate more diverse and novel recommendations. However, the number of candidate videos needed to be computed when generating recommendations is increased consequently. As a result, it will cost a longer time to produce recommendations, which may cause a recommendation delay to users, while long waiting time will decrease users' satisfaction. Therefore, in Tencent Video, we utilize another technique called the demographic based algorithm (DB) [25] to complement the recommendation results of matrix factorization (MF) based collaborative filtering (CF).

We first cluster users into different demographic groups according to their properties such as gender, age and education. Users in the same demographic group generally share similar interests or preferences. We compute the hot videos for each demographic group, i.e., the most popular videos. DB provides these hot videos as recommendations for a user based on his demographic group. By selectively merging the results of DB algorithm into the results of MF based CF algorithm, we broaden the span of recommendations and provide chances for users to discover new interests.

Demographic filtering has additional benefits. For new users or those users especially inactive who have little interaction with videos, the MF algorithm cannot produce enough efficient recommendations. In this case, we can rely on the demographic group information to better understand these users. Utilizing the results of DB algorithm, we can provide recommendation services for these users, so that the new user problem is partly solved. Specially, for new unregistered users, we generate the hot videos of global demographic group as recommendations, which was proven to work well in practice.

5.2.2 Demographic Training

Another optimization we utilized is demographic training, i.e., we run the recommendation algorithm within the afore-

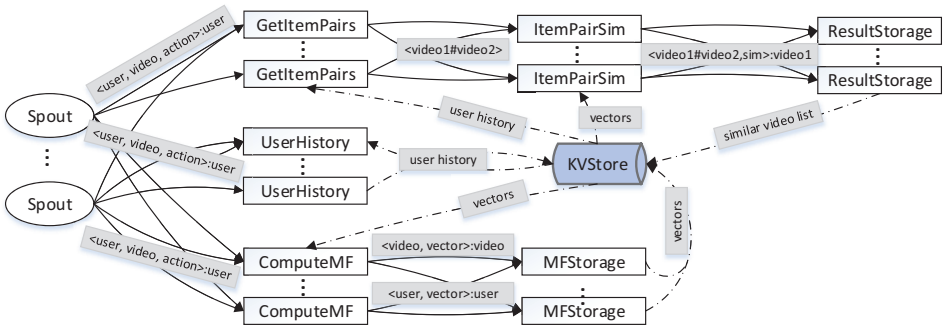


Figure 2: Topology Framework of the Recommendation System

mentioned demographic user groups. To be specific, there will be a video vector y_i for each demographic group, and the similarity between video pairs is computed within the demographic group.

The demographic training gives two benefits. On one hand, the user-video matrix of a demographic group is obviously less sparse than the global user-video matrix, as illustrated in [17]. We will get a more accurate model by running the recommendation algorithm within the demographic user groups. On the other hand, a more fine-grained model will be generated by demographic training, which can capture the variation of rating patterns between different demographic groups.

6. EXPERIMENTAL STUDY

Our video recommendation system is now deployed to provide recommendation services for Tencent Video, one of the largest video sites in China. The Storm cluster used to deploy is composed of 100 computers equipped with a four-core processor and 32 GB memory. Every day, it deals with billions of user action tuples, with data size of more than 1 TB. Nevertheless, it can provide accurate real-time video recommendations steadily, handling millions of user requests every day, with latency of milliseconds.

In this section, we discuss the effectiveness of our model by conducting comprehensive experiments on real data collecting from Tencent Video, and demonstrate its superior performance in production.

Parameters Setting: Parameters used in our model are determined by using grid search to obtain the optimal values, as shown in Table 2.

Table 2: Parameter Settings

	f	λ	a	b	η_0	α	β	ξ
Values	70	0.02	2.5	1	0.00001	1	0.02	10

6.1 Offline Experiments

To discuss our model in detail, we collect data from Tencent Video, and conduct experiments to compare the effect of the factors in model, including the effectiveness of demographic training and the effectiveness of adjustable updating strategy. We first briefly introduce the experimental settings for the evaluation, including dataset, and evaluation criterion.

Data Sets: We collect data from Tencent Video in one week, and reserve users who have more than 50 actions and videos with more than 50 related actions. We take the data of first six days as training data set and the last day as test data set. Statistics of the dataset after cleaning are shown in Table 3.

Table 3: DataSet Statistics

	Users	Videos	Actions	Test Actions
Counts	846928	18198	74503804	13838594

Evaluation Criterion: We use two evaluation metrics to measure and compare the performance of various models.

Traditionally, collaborative filtering algorithms are evaluated by the accuracy of their predicted ratings. One commonly used performance metric for rating accuracy is the Mean Absolute Error (MAE). However, we are interested in measuring the top-N recommendation performance instead of rating prediction. Furthermore, the reliable feedback regarding how much users like the videos, i.e., the true ratings, is not available. Thus the MAE measure is not appropriate in our condition. Instead of MAE, we measure the quality of recommendation models by looking at the *recall* metric, also known as hit rate, which is widely used for evaluating top-N recommendation systems.

The *recall* metric is defined as follows:

$$recall = \frac{\sum_{u \in U_{test}} \sum_{i_u} \ell(i_u \in \text{top-N}_u)/N}{|U_{test}|} \tag{13}$$

where i_u indicates that video i is liked by user u in test data and $\ell(z)$ is an indicator function that returns 1 if condition z holds, and 0 otherwise. A larger recall value indicates that the system is able to recommend more satisfactory videos, leading to a better performance.

Another evaluation metric we used is average rank. We denote $rank_{ui}$ as the percentile-ranking of video i within the ordered list of all videos recommended for user u, and $rank^t_{ui}$ as the percentile-ranking of video i within the ordered interested video list of user u in test data. Here, the ordered interested video list of user u is ranked by the corresponding user actions' confidence levels in testing data. This way, $rank_{ui} = 0\%$ means that video i is predicted to be the most desirable for user u while $rank_{ui} = 100\%$ indicates that video i is not recommended to user u, as the same case for $rank^t_{ui}$s. We choose to use percentile-ranks rather than absolute ratings in order to make our comparison general

and independent of the number of recommended videos, at the same time provide a more objective measure for implicit feedback. The average rank measure is then defined as the average percentile ranking of recommended videos, which is:

$$\overline{rank} = \frac{\sum_{u,i} rank_{ui}^t (1 - rank_{ui})}{\sum_{u,i}(1 - rank_{ui})} \quad (14)$$

where $1 - rank_{ui}$ measures the relative rating predicted by the model. For videos not recommended, $rank_{ui} = 1$, thus $1 - rank_{ui} = 0$. **Lower values** of $\overline{rank}$ is more desirable, indicating the recommended videos have an average lower rank in testing data, i.e., they lay closer to the top of actually watched video list.

6.1.1 *Effectiveness of Demographic Training*

In this section, we compare the models using demographic training and global training. Users in Tencent Video are clustered into dozens of groups, among which we select three largest demographic groups to do the comparison.

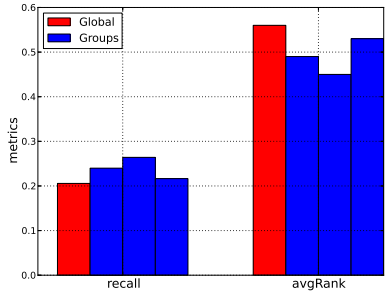


Figure 3: Comparison of Global vs Groups

The performance of global-model and group-models is shown in Figure 3, where the *recall* metric is shown by real numbers instead of percentage to make the values of $\overline{rank}$ (avgRank in the figure) and *recall* metric can be shown in the picture in an elegant way. From the figure, we can see that the *recall*s of group-models are higher than the global-model while the values of $\overline{rank}$ are lower than global-model, which suggests the performance of group-models is steadily superior to the global-model. The average improvement is more than 10%, while the maximum improvement is about 20%.

To explain the improvement of performance, we first show their statistics of training dataset in Table 4. The sparsity is defined by $\frac{\#Actions}{\#Users * \#Videos}$, which measures the quality of user-video matrix. Compared with the global dataset in Table 3 whose sparsity is 0.48%, the sparsity of dataset is greatly reduced within the demographic groups whose average sparsity is 1.45%. This is vital for the effectiveness of demographic training. Another reason why the group-models perform better should give the credit to the more fine-grained model generated by demographic training, which can capture the variation of rating patterns between different demographic groups.

6.1.2 *Effectiveness of Adjustable Updating Strategy*

In this section, we evaluate the effectiveness of our online adjustable updating strategy by comparing some alternative models.

Table 4: DataSet Statistics of Groups

	#Users	#Videos	#Actions	Sparsity(%)
Group1	166923	6594	14990035	1.36
Group2	152936	7161	13102179	1.20
Group3	95106	5010	8480328	1.78

First of all, we describe the alternative models as comparative methods.

- BinaryModel: This model uses the binary ratings but ignores the confidence levels of user actions, thus the learning rate is stable for all user action tuples.
- ConfModel: This model takes the confidence levels as ratings, and the learning rate is stable for all user action tuples.
- CombineModel: This model uses the binary ratings to train model and takes advantage of confidence levels of user actions to perform an adjustable updating, which is our ultimate recommendation model.

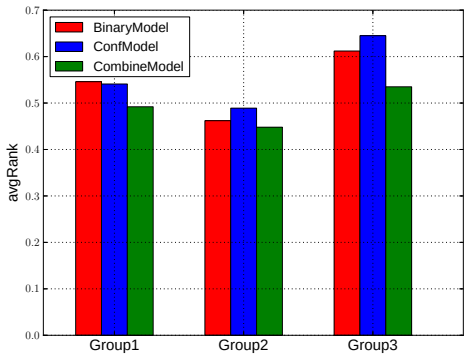
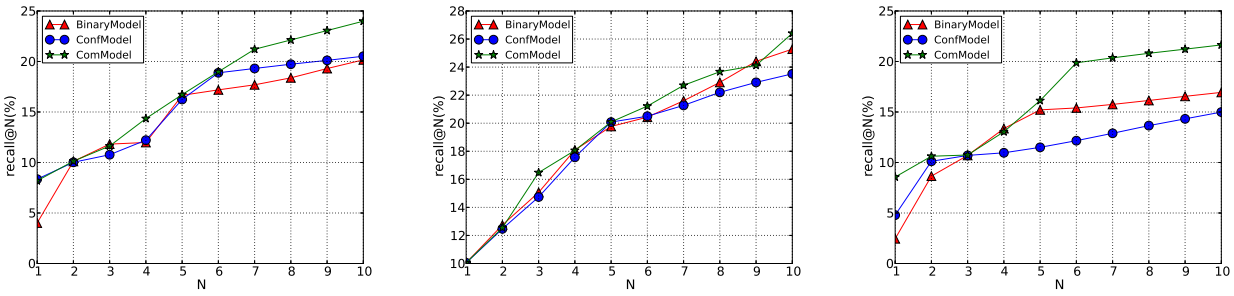


Figure 5: $\overline{rank}$ Comparison of Alternative Models

The performance of these models is shown in Figure 4 taking *recall*@N as metric, where N ranges from 1 to 10. We compare their performance in the three demographic groups. According to the figure, $CombineModel$ steadily performs better than the other two models, with an average improvement about 10%. Specifically, we arrive at the following two observations. First, the $BinaryModel$ performs slightly better than $ConfModel$ in most cases but not all. The reason why $ConfModel$ doesn't improve results is that it is sensitive to the settings in Table 1, which may raise noise due to inappropriate guess of implicit feedback. This verifies our statement in Section 3.2: if not appropriately handled, implicit feedback may reduce the performance of recommender systems, which is consistent with the results in [16]. Second, as we can see from the figure, $CombineModel$ has a much desirable performance than $BinaryModel$. This demonstrates the benefits of our adjustable training strategy. What's more, because we take advantage of the information of different user actions to measure the confidence levels of binary ratings, $CombineModel$ is more robust to the noisy data than $ConfModel$. As a result, $CombineModel$'s performance is much better than $ConfModel$.



(a) $recall@N$ of Group1 (b) $recall@N$ of Group2 (c) $recall@N$ of Group3

Figure 4: $recall@N$ Comparison of Alternative Models

Figure 5 compares alternative models' performances taking $\overline{rank}$ as metric. As expected, our $CombineModel$ has a significant superior performance with $\overline{rank}$ metric as well, and the performance of $BinaryModel$ is slightly better than $ConfModel$, as the same cases in Figure 4. As we can see, the values of $\overline{rank}$ metric of these models are around 0.5, indicating that the recommended videos rank in the middle of users' interested video list in average, which contributes to a considerable video recommendation list. Please note that the value of $\overline{rank}$ is not 0.5 for random model, since the total number of videos is much larger than the length of video recommendation list.

(a) Guess You Like Scenario

(b) Related Videos Scenario

Figure 6: Recommendation Scenarios

6.2 Online Testing

In this section, we show the practical performance of our recommendation system deployed in Tencent Video. The recommendation happens in two scenarios, as shown in Figure 6. The recommendations in Figure 6(b) are occurred when a user is watching a video, where we recommend similar videos of the current watching one. In the other scenario shown in Figure 6(a) where users open the site and have not watched any video yet, we first get the user history from KVStore, and then get the relevant videos of videos user liked before. Finally, video recommendations are generated by predicting user's preference for these videos utilizing the user vector and video vectors.

Comparative Methods: We compare our recommendation model with three competing methods that are used to produce video recommendations currently, as follows.

- Hot method recommends the most popular videos to users, a simple but powerful method, where the computation is in real-time.
- AR method represents the association rule based algorithm [28] whose model is trained in batch mode for every day.
- $SimHash$ method denotes the user-based CF algorithm utilizing the SimHash technique [4], which is also offline that the model is trained at regular intervals.
- rMF method refers to our real-time MF recommendation algorithm.

Evaluation Metric: To evaluate the video recommendation quality, we use click through rate (CTR) as the metric for online testing since it is objective and can be quickly computed.

Experimental Setup: We use live evaluation via A/B testing [10] as the main method for evaluating the performance of the recommendation system. To be specific, live traffic is diverted into distinct groups among which one group acts as baseline and others are exposed to other comparative recommendation models. These groups then can be compared against one another. This approach allows that evaluation takes place in the context of the actual site UI. We do the A/B testing for the comparative methods over a period of ten days and recording their CTRs for comparison.

Figure 7 shows the performances of the alternative methods taking CTR as metric. We eliminate the specific values of CTR in the figure due to the proprietary concerns. Still, we can conclude according to the figure that rMF performs better than other competing methods in most cases. The performance of Hot method is worst among these methods, while the AR and $SimHash$ have similar performances, and are better than Hot method. To better clarify the comparison, we show the performance improvement for different methods comparison in Table 5. From the table we can

Table 5: Performance Improvement Comparison

Day	1	2	3	4	5	6	7	8	9	10	**Average**
rMF vs AR (%)	13.23	7.73	7.68	7.49	2.88	9.83	17.83	2.08	1.94	17.06	**8.82**
rMF vs SimHash (%)	3.81	5.29	23.89	14.80	19.36	6.12	0.68	-12.72	-5.52	20.69	**7.04**
rMF vs Hot (%)	5.24	7.46	31.19	-0.80	27.91	7.20	30.40	-5.78	12.87	33.41	**14.72**

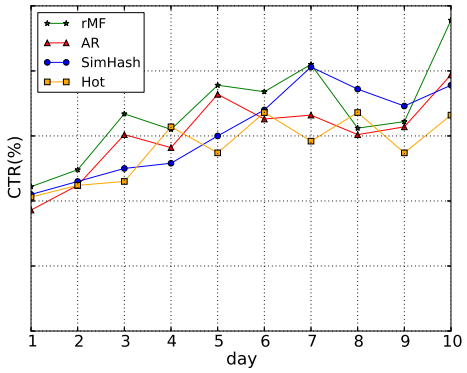


Figure 7: CTR Comparison in Online Testing

see that rMF has an average performance improvement of 8.82% and 7.04% comparing to AR and $SimHash$ methods respectively. Compared to the Hot method, rMF performs more superior with an average improvement of 14.72%. Therefore, it is clearly verified that our recommendation system is effective in providing real-time accurate recommendation service. Note that, such improvement is already very valuable in real business applications. In addition, although rMF has a better performance in most cases, there exist some cases where it performs slightly poor. Actually, all the competing methods have both good cases and bad cases where their performances are excellent or poor, as the user behaviors may be affected by real-world events and are not fully predictable. For example, a user may have different preferences in different situations, depending upon his mood or with whom he's watching the video. The good and bad cases indicate that each method has its advantages and disadvantages, so that it may be better to combine results from different methods in an efficient way.

7. CONCLUSION AND FUTURE WORK

In this paper, we proposed a real-time top-N video recommendation system. First, we developed a real-time MF based CF algorithm, with an online updating strategy proposed. Our online updating strategy took the confidence levels of different implicit feedback data instances into consideration, and the learning rate of the training process is adjustable to each data instance. Second, we introduced the procedure of real-time video recommendation generation. To produce accurate video recommendations in real-time, we built similar video tables to select high-quality candidate videos for recommendation, recording the most relevant videos for each video. We leveraged factors from three aspects to compute the similarity of videos, including the matrix factorization similarity, the type similarity and the time factor. Third, we presented how to implement the model scalably on Storm, a distributed real-time computation system for processing streams of data. In addition, some optimization techniques in production were proposed, including demographic filtering and demographic training. At last, we conducted comprehensive experiments to evaluate our model, including offline experiments and online testing. In offline experiments, we collected real data from Tencent Video and conducted experiences to discuss the effect of factors in our model, including the effectiveness of demographic training, and effectiveness of adjustable training. Our video recommendation system was in production at Tencent Video and dealt with billions of user action tuples everyday, with data size of more than 1 TB. We show its superior performance in production in the online testing experiments.

There are a series of future works we can do for real-time video recommendations. First, we only utilized the videos' type information to measure the content features of videos in this work. There are still some other useful information can be employed, such as the videos' directors and actors. Second, there still remains much improving space for enhancing the diversity and novelty of video recommendations. A possible improvement may be a better methodology for combining results from different algorithms to leverage benefits of different methods.

8. ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China under Grant No. 61272155 and 61572039, 973 program under No. 2014CB340405, Beijing Natural Science Foundation (4152023), Shenzhen Gov Research Project JCYJ20151014093505032, and Tecent Research Grant (P-KU).

We thank all the Tencent Recommender Platform Team members for their contributions to this paper. Many thanks to Yu Xiang, Yong Wei, Qiaoxiong Lin, and Ruhai Luo for their kind suggestions and support.

9. REFERENCES

[1] J. Abernethy, K. Canini, J. Langford, and A. Simma. Online collaborative filtering. *University of California at Berkeley, Tech. Rep*, 2007.

[2] R. M. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Proc of the 2007 7th IEEE ICDM Conference*, pages 43–52. IEEE, 2007.

[3] B. Chandramouli, J. J. Levandoski, A. Eldawy, and M. F. Mokbel. Streamrec: A real-time recommender system. In *Proc of the 2011 ACM SIGMOD Conference*, pages 1243–1246, 2011.

[4] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proc of the 34th ACM STOC Conference*, pages 380–388. ACM, 2002.

[5] B. Chen, J. Wang, Q. Huang, and T. Mei. Personalized video recommendation through tripartite graph propagation. In *Proc of the 20th ACM ICMM Conference*, pages 1133–1136. ACM, 2012.

[6] C. Chen, H. Yin, J. Yao, and B. Cui. Terec: A temporal recommender system over tweet stream. *Proc. VLDB Endow.*, 6(12):1254–1257, Aug. 2013.

[7] B. Cui, J. Jiang, Q. Huang, Y. Xu, Y. Gui, and W. Zhang. Pos: A high-level system to simplify real-time stream application development on storm. *Data Science and Engineering*, pages 1–10, 2015.

[8] B. Cui, H. Mei, and B. C. Ooi. Big data: the driver for innovation in databases. *National Science Review*, 1(1):27–30, 2014.

[9] P. Cui, Z. Wang, and Z. Su. What videos are similar with you?: Learning a common attributed representation for video recommendation. In *Proc of the 2014 ACM ICMM*, pages 597–606. ACM, 2014.

[10] J. Davidson, B. Liebald, J. Liu, et al. The youtube video recommendation system. In *Proc of the 4th ACM RecSys Conference*, pages 293–296. ACM, 2010.

[11] Z. Deng, M. Yan, J. Sang, and C. Xu. Twitter is faster: Personalized time-aware video recommendation from twitter to youtube. *TOMM*, 11(2):31, 2015.

[12] E. Diaz-Aviles, L. Drumond, Z. Gantner, et al. What is happening right now... that interests me?: online topic discovery and recommendation in twitter. In *Proc of the 21st ACM CIKM Conference*, pages 1592–1596. ACM, 2012.

[13] E. Diaz-Aviles, L. Drumond, L. Schmidt-Thieme, and W. Nejdl. Real-time top-n recommendation in social streams. In *Proc of the 6th ACM RecSys Conference*, pages 59–66, 2012.

[14] S. Gao, D. Zhang, H. Zhang, C. Huang, Y. Zhang, J. Liao, and J. Guo. Veclp: A realtime video recommendation system for live tv programs. In *Proc of the 29th AAAI Conference*, 2015.

[15] T. George and S. Merugu. A scalable collaborative filtering framework based on co-clustering. In *Proc of the 5th IEEE ICDM Conference*, pages 4–pp. IEEE, 2005.

[16] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Proc of the 2008 8th IEEE ICDM Conference*, pages 263–272, 2008.

[17] Y. Huang, B. Cui, W. Zhang, J. Jiang, and Y. Xu. Tencentrec: Real-time stream recommendation in practice. In *Proc of the 2015 ACM SIGMOD Conference*, pages 227–238. ACM, 2015.

[18] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proc of the 14th ACM SIGKDD Conference*, pages 426–434. ACM, 2008.

[19] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[20] G. Ling, H. Yang, I. King, and M. R. Lyu. Online learning for collaborative filtering. In *Proc of the 2012 IEEE IJCNN Conference*, pages 1–8. IEEE, 2012.

[21] T. Mei, B. Yang, X.-S. Hua, and S. Li. Contextual video recommendation by multimodal relevance and user feedback. *TOIS*, 29(2):10, 2011.

[22] M. Papagelis, I. Rousidis, D. Plexousakis, and E. Theoharopoulos. Incremental collaborative filtering for highly-scalable recommendation algorithms. In *Foundations of Intelligent Systems*, pages 553–561. Springer, 2005.

[23] J. Park, S.-J. Lee, S.-J. Lee, et al. Online video recommendation through tag-cloud aggregation. *IEEE MultiMedia*, 1(18):78–87, 2011.

[24] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proc of 2007 KDD cup and workshop*, volume 2007, pages 5–8, 2007.

[25] M. J. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artif. Intell. Rev.*, 13(5-6):393–408, Dec. 1999.

[26] R. Pereira, H. Lopes, K. Breitman, V. Mundim, and W. Peixoto. Cloud based real-time collaborative filtering for item–item recommendations. *Computers in Industry*, 65(2):279–290, 2014.

[27] S. Rendle and L. Schmidt-Thieme. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *Proc of the 2008 ACM RecSys Conference*, pages 251–258, 2008.

[28] J. J. Sandvig, B. Mobasher, and R. Burke. Robustness of collaborative recommendation based on association rule mining. In *Proc of the 2007 ACM RecSys Conference*, pages 105–112, 2007.

[29] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Proc of the 5th IEEE ICIS Conference*, pages 27–28. IEEE, 2002.

[30] A. Toshniwal, S. Taneja, A. Shukla, et al. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156. ACM, 2014.

[31] J. Vinagre, A. M. Jorge, and J. Gama. Fast incremental matrix factorization for recommendation with positive-only feedback. In *User Modeling, Adaptation, and Personalization*, pages 459–470. Springer, 2014.

[32] Z. Wang, Q. Li, Y. Liu, W. Liu, and J. Yin. Online personalized recommendation based on streaming implicit user feedback. In *Web Technologies and Applications*, pages 720–731. Springer, 2015.

[33] B. Yang, T. Mei, X.-S. Hua, et al. Online video recommendation based on multimodal fusion and relevance feedback. In *Proc of the 6th ACM CIVR Conference*, pages 73–80. ACM, 2007.

[34] H. Yin, B. Cui, L. Chen, Z. Hu, and Z. Huang. A temporal context-aware model for user behavior modeling in social media systems. In *Proc of the 2014 ACM SIGMOD Conference*, pages 1543–1554, 2014.

[35] H. Yin, B. Cui, Y. Sun, Z. Hu, and L. Chen. Lcars: A spatial item recommender system. *ACM Trans. Inf. Syst.*, 32(3):11:1–11:37, July 2014.

[36] L. Yu, Y. Shao, and B. Cui. Exploiting matrix dependency for efficient distributed matrix computation. In *Proc of the 2015 ACM SIGMOD Conference*, pages 93–105. ACM, 2015.

[37] X. Zhou, L. Chen, Y. Zhang, et al. Online video recommendation in sharing community. In *Proc of the 2015 ACM SIGMOD Conference*, pages 1645–1656. ACM, 2015.