A Recommender System for Heterogeneous and Time Sensitive Environment

Meng Wu, Ying Zhu, Qilian Yu, Bhargav Rajendra, Yunqi Zhao, Navid Aghdaie, and Kazi A. Zaman
Electronic Arts, Inc.
Redwood City, CA, USA

ABSTRACT

The digital game industry has recently adopted recommender systems to deliver the most relevant content and suggest the most suitable activities to players. Because of diverse game designs and dynamic experiences, recommender systems typically operate in highly heterogeneous and time-sensitive environments. In this paper, we describe a recommender system at a digital game company which aims to provide recommendations for a large variety of use-cases while being easy to integrate and operate. The system leverages a unified data platform, standardized context and tracking data pipelines, robust naive linear contextual multi-armed bandit algorithms, and experimentation platform for extensibility as well as flexibility. Several games and applications have successfully launched with the recommender system and have achieved significant improvements.

CCS CONCEPTS

• **Information systems → Retrieval models and ranking**; *Information integration*; *Evaluation of retrieval results.*

KEYWORDS

Recommender System; Digital Game; Multi-armed Bandits; Feature Embedding

ACM Reference Format:
Meng Wu, Ying Zhu, Qilian Yu, Bhargav Rajendra, Yunqi Zhao, Navid Aghdaie, and Kazi A. Zaman. 2019. A Recommender System for Heterogeneous and Time Sensitive Environment. In *Thirteenth ACM Conference on Recommender Systems (RecSys '19), September 16–20, 2019, Copenhagen, Denmark.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3298689.3347039

1 INTRODUCTION

Recommender systems that deliver relevant and personalized contents to users have been widely utilized by video and music streaming providers, news websites, advertisement and e-commerce companies [3, 11, 17, 25, 36]. Although recommender systems are relatively new to the digital gaming industry, several recent studies on recommending suitable games and contents to players have shown promising results [18, 29]. Recommendation applications in the digital game industry are not limited to promoting game sales. The system can also be used to suggest the next in-game activity and optimize the player experience. To name a few, the activities can be tutorials to watch, skill games to practice, maps to play, or squads to join. While applying the recommender system to the digital games, some new challenges to both system and algorithm designs have been brought up to attention.

First of all, similar to e-commerce and video websites, recommender system can suggest the next favorite games to players in game stores and subscription service [18, 29]. The recommender system analyzes game genres, ratings, as well as past gameplay history to fit the taste of players. The second recommendation application is to optimize player journey inside games by suggesting game modes, maps, roles, and objectives. Many games contain different maps and game modes to provide a diverse experience. Different options often require different skill levels, strategies, or cooperations between players. Helping players to join the most suitable gameplay is essential for their onboarding experience and engagement.

Other examples of recommended in-game contents may include tutorials, hints, news, or user-generated videos. Each content type influences the player experience from a different perspective. For example, tutorials and hints are indispensable to new players to understand the basics of the game and overcome unnecessary barriers. Skill games help players to practice and achieve better performance in real competitions against others. Last but not least, players naturally form a well-connected community. Many people enjoy learning and watching others' strategies and techniques. Promoting good user-generated contents helps to build an active community among players.

Not surprisingly, the new opportunities in digital games bring new challenges to the system and algorithm designs. We summarize the challenges as the recommender system has to work in a highly heterogeneous and time-sensitive environment. The heterogeneous environment is caused by: a) diversity of the game designs; b) different gameplay experiences. Digital games fall into a wide range of genres such as first player shooting (FPS), sports, and role-playing games (RPG) [1]. Different game genres offer completely different gameplay mechanisms and experiences. Even within the same genre, games are typically designed to be unique in both concept and context with varied contents, gameplay, progression, and social systems. On the other hand, players usually have different play styles, such as strategic leaders, story writers, and competitive gamers, which affect the enjoyment of the contents. Player's skills and interests in a game may change rapidly during the gameplay and progression.

The second major challenge for the recommender system is that digital games are also very time-sensitive environments. Figure 1 shows an example of the number of new and active users in days

RecSys '19, September 16–20, 2019, Copenhagen, Denmark

ACM ISBN 978-1-4503-6243-6/19/09…$15.00
https://doi.org/10.1145/3298689.3347039

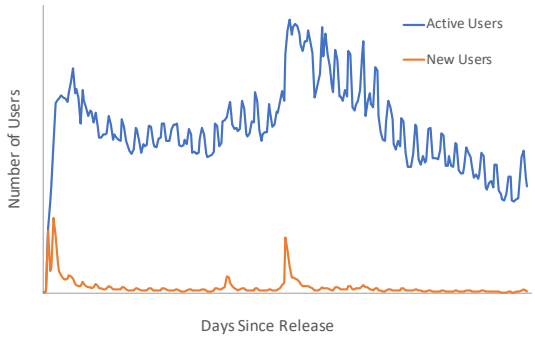

Figure 1: Example of the number of new players and active users in each day after the game release.

after the game release. Most of the players joined the game in the first few weeks after the release. There are several peaks of new players joining because of the holiday sales and new contents added to the games. Every time a game launches, the recommender system always faces a cold start problem. The players are also progressing in the game and never return to the "level zero". Therefore, the recommender system need to learn from the first day of the game release as quickly as possible, then continues to adapt to the changes in the game. From Figure 1, the start date could provide an essential signal on player engagement pattern. A recommender system behaving differently for different cohorts can potentially improve the performance, which also adds complexity to both of the challenges.

The game industry has transformed from producing ship-and-forget games to providing 24/7 high availability gaming platforms. While the game is continuously adding contents and live events, the recommendation system must stay up-to-date. Our company launches tens of new games every year. Every game desires to use a recommender system to provide personalized and optimized experiences. Building separated systems prevent recommendation applications from scaling up and coordinating well. Therefore, we developed one centralized recommender system to serve many games under the highly heterogeneous and time-sensitive environment. The centralized system [1] includes unified data collection, machine learning, model execution, and realtime performance measurement via experiments.

2 RELATED WORKS

There have been recent interests in analyzing the player preferences to build recommender systems for games, but less mature than many other industries. Players' behaviors within the games have been studied from many perspectives to improve the current game or a new game design [19]. It is a general belief that players tend to like related or alike games [29]. Meidl et al. studied co-clustering of the adjective-context word pairs in the game reviews to find the similar games [26]. Several approaches applied the collaborative filtering algorithms on the individual preferences and ratings to discover the similarity between gamers [9, 30]. Chow et al. developed a novel hybrid algorithm, HybridRank, to deliver recommendations for mobile games and implemented in a live mobile game platform [14]. The HybridRank algorithm utilizes a personalized random walk algorithm to incorporate both content-based and user-based information.

Looking beyond the game industry, the recommender system researches have advanced in many dimensions [2, 7]. For our purposes, we will focus on three axes: width, depth, and breath. Horizontally scaling for width usually means the system has to be able to process extensive data set, find related content from huge pools, and provide service with high throughputs. Scaling in this dimension is essential for many companies like the e-commerce and user-generated content provider [16, 31, 38]. It requires science and engineering to work hand in hand to build highly optimized searching and ranking systems. We reserve the dimension of depth to capture efforts to build recommendation models. A recommendation model can be as simple as market basket rule or trending order [28]. More depth in algorithms can be a well-tuned collaborated filtering and hybrid approaches [15]. Many recommendation algorithms also benefit from the deep learning research to leverage massive neural network [16, 20]. The final dimension, breadth, draws attention to the number of the applications themselves. The design considerations for a platform capable of supporting a few tasks are different from supporting hundreds. Recently, several cloud service vendors started to provide recommendation solutions based on their cloud infrastructure and services [6, 8]. Customers only need to build and maintain a few components like data ingestion and web services to have an end-to-end recommender system.

3 SYSTEM ARCHITECTURE

Many recommender systems contain dedicated data pipelines and tailored model training and model execution services to achieve the best results. In order to support various logics and requirements from many games, our design principle is to identify repeatable processes in data processing and modeling procedure, and then generalize and automate via a centralized platform. A high-level architectural view of the data platform and recommender system appears in Figure 2. The recommender system is built on top of the centralized data warehouse which provides complete data services to all games. The data service includes standardized telemetry, data extract, transform, load (ETL) pipelines, ad-hoc analysis, machine learning environment, experimentation, and the recommendation components.

Our system abstracts each recommendation as a function, taking a set of candidate items as input, and outputting the best item(s) that leads to a better experience. External entities, such as game clients and servers, retrieve the real-time recommendation through remote procedure call (RPC) to the recommendation execution service. The system integration includes the recommendation API that receives the inputs and responses with the outputs, and telemetry data to capture recommendation feedbacks.

[1] Collection, processing, storage, and use of data, as well as personalization models, always honor relevant legal requirements, user account controls, and privacy settings. Please review company official disclosures and policies at https://www.ea.com/legal for more details.

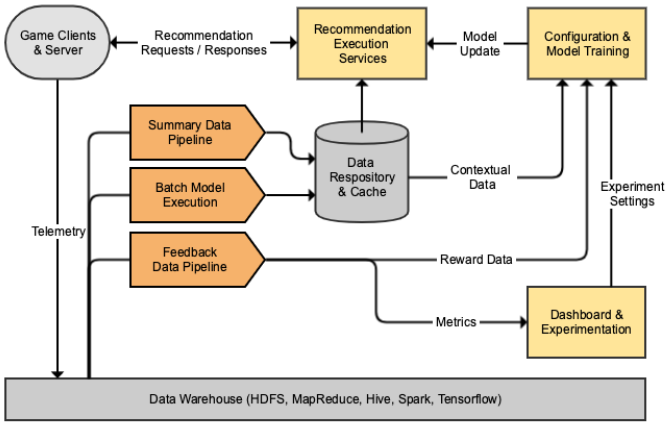

Figure 2: Block diagram of recommender system components embedded within the data platform. The recommender system leverages unified data processing, machine learning and experimentation systems.

3.1 Candidate Generation

In the candidate generation process, we focus on providing flexible ways for storage and retrieval logic for various applications. The recommender system offers two ways to retrieve possible candidates as the input. The first way is standard metadata storage that provides generic management operations. The storage component constructs indexing and allows to fetch for candidate items based on predefined triggers. The second way is to send the candidate items through the recommendation RPC request to let the game client handle the custom logic. Both approaches standardize the candidate items to the same format as the input of the live recommender service.

3.2 Execution Service

The recommendation execution component processes the request and executes pre-defined rules and models as a live service. All recommendation execution processes are formulated as a collection of condition-action rules with dynamic priorities, as shown in Algorithm 1. The condition qualifies if one recommendation rule/model should proceed for a given context and experiment setting. The action contains custom logic and model execution to generate recommendation results. The prioritization provides a way to resolve conflicts when multiple results exist. The conditions, actions, and prioritization are established using an interpreting programming language, JRuby [27], in a serverless fashion. The level of flexibility by the script execution has dramatically decreased the development time from weeks to an afternoon for many applications.

3.3 Data Pipeline

The data warehouse at the bottom of Figure 2 serves as the foundation for data storage and computing. The recommender system leverages three types of data processes inside the data warehouse.

(1) Data ETL processing pipeline to generate the aggregated game historical data.

Algorithm 1: Rule-based Execution Paradigm

```
Function recommend (candidates, context ):
Optional preprocessing codes ...
results = []
forall Registered rules do
    if rule.condition( context ) then
        Optional rule action codes ...
        ranked_items = model( candidates, context )
        results.add( ranked_items, rule.priority(context) )
    end
end
Optional postprocessing codes ...
Return result with the highest priority
```

(2) Batch execution of machine learning models to pre-compute results and intermediate features.
(3) Telemetry extraction to track user feedbacks.

The first type of data contains our repository of anonymized player data that collected via standard telemetry from game clients and servers. The warehouse uses the Hadoop eco-system for processing and storage. The telemetry data are processed via MapReduce, Hive SQL, Storm, and Spark to standard summary tables. The data from multiple sources are stored together to provide a holistic understanding of our games. The anonymized data and aggregations are standardized for recommender model training and inference. The same data are offered as batch tables in the Hadoop cluster and are stored in the NoSQL database for real-time usage.

Besides the conventional ETL data processes, the data warehouse also executes machine learning models in batch fashion. The machine learning models such as collaborative filtering[2] and predictive analytics models directly compute the recommendation results to store in the cache. There are machine learning algorithms that can also provide intermedia features and embedded contexts from the telemetry data. Those machine learning models enable us to transfer high dimensional and complex features into low dimensional latent/embedded features. Those dense features are then adapted to specific recommendation applications by the online learning algorithm, which will be discussed in the later section of this paper. To maximize the reusability of machine learning models, we usually apply the batch execution to compute the latent/embedded features instead of the direct results.

Feedbacks of recommendations are parts of standard telemetry and processing pipelines of the data platform. The feedback data are used as the optimization objectives during the machine learning process. The recommender system is required to optimize both short-term and long-term goals for different applications. Hence, we build the feedback data pipeline to include short-term effects such as impression, click, and conversion as well as the long-term feedbacks such as completing the match, and progress to the next level. Different feedback types are merged into an anonymized format as `(session_hashcode, application_id, item_id, feedback_type, feedback_value, timestamp)`. The standard format facilitates the model training processes to optimize any of

[2]https://spark.apache.org/docs/latest/mllib-collaborative-filtering.html

the feedback types via simple configurations. Note that, for long-term feedback such as the second-day retention, there is at least a 24 hours delay in data availability. Thus, it is necessary for the recommender system to be able to combine different types of feedbacks and react based on hybrid objectives.

3.4 Experimentation

Recommendation models must be measured by their impact rather than accuracy. Optimizing the impact requires a closed-loop data system that can trace immediate feedbacks such as click on an item or enter a match, also longer-term behaviors such as progression and retention. The experimentation service automatically measures the aggregated effects from the feedback data pipeline, which is also used for model training. A recommendation, in the context of an experiment, is achieved by linking the condition qualification of the rule in Algorithm 1 to any randomized experiment group. The seamless integration between recommendation and experimentation allows us to measure overall performance across a wide range of predefined performance metrics. Experiments help us identify the best model and ensure changes to our recommender system have a real positive impact.

4 ALGORITHMS

When facing the cold start and time-sensitive conditions, it is natural to utilize reinforcement learning algorithms with prepared context and feedback data [39]. The MAB algorithms have demonstrated promising strength in many recommendation applications [4, 5, 23]. Our recommender system has adopted the multi-armed bandit (MAB) algorithms as the self-learning and optimization models for many applications. The MAB algorithms typically solve a cold start problem by balancing the exploration and exploitation to minimize the total regrets. The algorithms have been extended to not only base on the organic rewards but also incorporate contextual information with machine learning estimators.

The contextual MAB models in our system can utilize any available contextual information from the data pipeline and batch model execution, and the standardized feedback data as the rewards. The contextual MAB algorithms such as LinREL/LinUCB algorithm uses the ridge regression to predict rewards and select the next arm under the upper confidence bound (UCB) framework [10, 22, 24]. The linear models can be combined with randomized strategies such as Thompson Sampling which is often more robust with delayed or batched feedbacks in practice [12]. In our scenarios, the contextual MAB algorithms need to solve several practical challenges due to the heterogeneous and time-sensitive environment without any human supervision.

(1) In practice, the input data are often block-wise missed, since it is common that someone has never played some games and all features related to those games are not available. It is challenging for the conventional missing data handling techniques to resolve a large portion of block-wise missed values, especially with the cold-star problems.
(2) When a game continuously evolves and develops, it is also common to have new features added to the model. The online learning algorithm should support plug in and out features with small efforts to update.

Algorithm 2: Linear Contextual MAB Algorithm

forall *t = 1, 2, 3, ..., T* **do**
  forall *arms a* **do**
    Generate score s_a by sampling from (2).
  end
  Choose arm $a_t = \text{argmax}\ s_a$.
  Observe a real-valued payoff $r_{t,a}$ and update (3) and (4).
end

(3) The algorithm needs to be robust to high dimensional features or be able to automatically performing feature selection. Many contextual MAB algorithms use the matrix inversion in the linear model, which could be an ill-posed inversion problem when the high dimensional features are correlated.
(4) It is desired to extend beyond the linear model to benefit from other non-linear methods.

This section begins with introducing a typical linear contextual MAB algorithm with the Thompson Sampling, and then describes our proposed modifications to solve the first three challenges. The last part of this section presents a simple way to combine the MAB algorithm with other non-linear methods.

4.1 Contextual MAB

The contextual MAB algorithm in our recommender system applies a combination of the linear pay-off models with Thompson Sampling. The benefit of the linear model with Gaussian assumption allows simple additive properties in data aggregation and model combination [24]. The linear pay-off models assume the reward $r_{t,a}$ of arm a is linear with the input features x_t at trial/time t

$$\mathbf{E}[r_{t,a}|x_t] = \eta_{a,t}^T x_t, \tag{1}$$

where $\eta_{a,t} \in \mathbf{R}^{d+1}$ denotes the linear weights, and $x_t \in \mathbf{R}^{d+1}$ is the feature vector with intercepter i.g.

$$x_t = [x_{t,1}, x_{t,2}, ..., x_{t,d}, 1]^T.$$

The estimation of the average reward based on the historical pull c_t of arm a approximates as a Gaussian distribution

$$\hat{r}_{t,a}(x_t) \sim \mathcal{N}(\mu_{t,a}, \sigma_a^2(x_t^T A_{t,a}^{-1} x_t)), \tag{2}$$

where the mean $\mu_{t,a} = (A_{t,a}^{-1} b_{t,a})^T x_t$ with

$$A_{a,t} = \sum_{c_{t'}=a \forall t' < t} x_{t'} x_{t'}^T + \rho \mathcal{I}, \tag{3}$$

and

$$b_{a,t} = \sum_{c_{t'}=a \forall t' < t} r_{t',a} \cdot x_{t'}, \tag{4}$$

where ρ is a positive constant to prevent ill-posed problem and form a ridge regression. When the contextual features do not span the full linear space, the estimator's prediction confidence interval in Eqn. (2) will not decrease with more observations, if the $\mathbf{E}[A_a]$ is not a full rank matrix. Thus, the multi-armed bandit model may not converge to the best arm because it is easy to show that $\lim_{t\to\infty} ||\mathbf{E}[A_{a,t}]^{-1}|| = \frac{1}{\rho}$. Note that, σ_a is the standard deviation of the rewards of arm a alone. Using constant values or linear to the total regrets are both reasonable choices [5, 24].

4.2 Naive Linear Contextual MAB

A common way to simplify the high dimensionality plus sparsity problem is to introduce extra assumption like Naive Bayes. Similar to the Naive Bayes classifier that is commonly used in the text classification problem, we assume each feature variables are independent. The reward estimator is then a linear combination of the estimators by the individual feature variables. To formula the assumptions mathematically,

$$\mathbf{P}(x_{t,i}; x_{t,j}) = \mathbf{P}(x_{t,i})\mathbf{P}(x_{t,j}) \quad \text{if } i \neq j, \tag{5}$$

and

$$\begin{aligned}\mathbf{E}[r_{t,a}|x_t] &= \sum_{i=1}^{d} w_i \cdot \mathbf{E}[r_{t,a,i}|x_{t,i}] = \sum_{i=1}^{d} w_i \cdot \mu_{t,a,i} \\ &= \sum_{i=1}^{d} w_i(\eta_{t,a,i}^T \cdot [x_{t,i}, 1]),\end{aligned}$$

where w_i denotes the combination weighting, $\mu_{t,a,i}$ is the linear prediction only using the i-th feature, and $\eta_{t,a,i} \in \mathbf{R}^2$ is the coefficient of the simple linear models. From each feature variable, we can obtain estimator with distribution of

$$\tilde{r}_{t,a,i}(x_{t,i}) \sim \mathcal{N}(\mu_{t,a,i}, \sigma_a^2(\frac{1}{n_{a,i}} + \frac{x_{t,i}^2}{\sum_{c_{t'}=a \forall t' < t} x_{t',a,i}^2})), \tag{6}$$

where $n_{a,i}$ is the number of times features i recorded when armed a is selected. To obtain the final estimator for the Thompson Sampling, we need to merge the means and variances from all the input features. Because the linear estimators $\mu_{t,a,i}$ are unbiased estimator, simply weighted average still maintains the unbiased property and Gaussian distribution as

$$\tilde{r}_{t,a}(\tilde{r}_{t,a,1}, \tilde{r}_{t,a,2}, \ldots, \tilde{r}_{t,a,d}) = \frac{\sum_i w_i \cdot \tilde{r}_{t,a,i}}{\sum_i w_i}, \tag{7}$$

where w_i denotes the weight for each estimator with $w_i \geq 0 \forall i$. There are two strategies to determine the weights:

1. Simply use one for all the weights, when assuming all the features have equivalent confidences of the estimation of the reward.
2. Use the inverse of the standard deviation in Eqn. (6) as the weight. This weight could compensate the confidence differences but may be influenced by the sample noise.

To obtain a reasonable variance, we propose to match the combined variance from Eqn. (6) with full linear model's variance in Eqn. (2) under the independent assumption in (5). Thus, the variance of the combined models is

$$\tilde{\sigma}_{t,a}^2 = \sigma_{t,a}^2(\frac{1}{n_a} + \sum_i \frac{x_{t,i}^2}{\sum_{c_{t'}=a \forall t' < t} x_{t',a,i}^2}). \tag{8}$$

Algorithm 3 summarizes the proposed naive linear contextual MAB algorithm. In the algorithm, μ and σ_a^2 are the baseline Gaussian approximation parameters of the reward of the arm a without any contextual information, and ϵ denotes the rate of exploring new items. We generalized Naive Bayes assumption to each available group of the features that provides a linear estimation of the reward. Then the final estimation of the reward $\tilde{\mu}_a$ and standard deviation $\tilde{\sigma}_a^2$ combines the estimators from all feature groups. The

Algorithm 3: Naive Linear Contextual MAB Algorithm

```
forall t = 1, 2, 3, ..., T do
    Observe feature groups x_i ∈ R^{n_i} of the user
    forall arm a ∈ A_t do
        if aisnew then
            Explore arm a with rate ε
            μ_{t,a} ← 0 , σ²_{t,a} ← 1 , n_a ← 0
        else
            forall Available feature groups x_{t,i} do
                if A_{a,i}, b_{a,i} not exist then
                    A_{a,i} ← ρI_{n_t}
                    b_{a,i} ← 0_{n_t × 1}
                end
                μ̃_{a,i} ← (A^{-1}_{a,i} b_{a,i})^T x_{t,i}
                σ̃²_{a,i} ← x^T_{t,i} A^{-1}_{a,i} x_{t,i}
            end
            Combine estimator mean and variances based on
             (7) and (8).
            Generate scores for arm a: s_a ~ N(μ̃_a, σ̃²_a).
        end
    end
    Choose arm a_t = argmax s_a, with ties broken arbitrarily.
    Observe a real-valued payoff r_{a,t}.
     μ_{t+1,a} ← (r_{t,a} + μ_{t,a} · n_a)/(n_a + 1)
    σ²_{t+1,a} ← ((r_{t,a} − μ_{t,a})² + σ²_{t,a} · n_a)/(n_a + 1)
    n_a ← n_a + 1
    forall Available feature groups x_i do
        A_{a,i} ← A_{a,i} + x_{t,i} x^T_{t,i}
        b_{a,i} ← b_{a,i} + r_{t,a} x_{t,i}
    end
end
```

MAB algorithm applies Thompson Sampling process to balance the exploration and exploitation.

There are several advantages with the Naive linear model when the running multi-armed bandit algorithms as a live service. First, the missing data is no longer an issue in the model training and inference. Secondly, any newly developed feature can be added or removed from the models without resetting the existing trained models. This property gives us great flexibility to improve the feature engineering anytime without worrying about the interruption to the live service. Also, the automatic feature selection becomes possible because the features are used separately. To make the feature selection, we can calculate the coefficient of determination (R^2) for each feature. During the reward prediction, the algorithm skips the feature if the R^2 score is below a certain threshold.

To work in the time-sensitive environment, the MAB algorithm should include temporal information by default [22]. In our everyday use cases, the temporal change can be periodic, also known as seasonality, and aperiodic. For the seasonality, we use one hot encoded time of date and day of the week as the features for the naive linear contextual MAB algorithms. For the aperiodic changes, the Kalman filter and particle learning can be applied to adjust the learned models dynamically [37]. Sudden changes detection

mechanism can also be applied to handle aperiodic changes. In practice, we found out that adding a data retention or an exponential temporal decay works effectively.

4.3 Hybrid Contextual MAB

In recent years, there have been several studies to marry the MAB algorithms with the collaborative filtering methods [21, 33]. One of the difficulties is the high computational cost of updating collaborative models in online learning. Furthermore, the newly collected feedbacks may require different hyperparameters from the cold start to the state with massive data points. However, if we treat the collaborative filtering as a feature embedding methods for the contextual MAB layer instead of an end-to-end estimator, the time and computing complexity become less critical. In our design, we want to leverage the flexibility of the naive contextual model by utilizing the collaborative filtering method as one feature engineering source.

Let us take the latent factor model as an example. The latent factor model uses the low-rank matrix approximation of the user-item relationship matrix as

$$\text{minimize} \sum_{i,j} I_{i,j} \|M_{i,j} - V_i U_j^T\|_2^2 + \lambda_1 \sum_i \|V_i\|_2^2 + \lambda_2 \sum_j \|U_j\|_2^2, \tag{9}$$

where M denote the user-item relationship matrix, I is an indicator of whether to use the entry. Vectors $V_i \in \mathbf{R}^k$ and $U_j \in \mathbf{R}^k$ are the latent features of the item i and the user j, k is the rank of the approximation, and λ's denote weights to balance different parts in the objective function. The pre-computed latent features of the users can be used as one of the contextual feature groups in Algorithm 3. If we consider the data in the user-item relationship matrix as the historical replays for the linear contextual MAB algorithm. Then the linear estimator parameters $\eta_i^{CF} \in \mathbf{R^{k+1}}$ for item i using the CF features are obtained by

$$\text{minimize} \sum_j I_{i,j} \|M_{i,j} - [U_j, 1]^T \eta_i^{CF}\|_2^2 + \rho \|\eta_i^{CF}\|_2^2. \tag{10}$$

In many cases, the elements in the latent features are almost linearly independents, because the connection to the low-rank singular value decomposition. Therefore, the elements in the latent features can be used separately to construct the naive linear estimators.

The idea of using the collaborative filtering method to generate the latent features with small dimension is very similar to an embedding process. The multilayer perceptrons can also be applied to the matrix decomposition task to generate non-linear latent factors for prediction [38]. The latent/embedded features of the users or items are computed by the batch model execution pipeline and stored together with the regular features. The contextual MAB algorithm adapts the machine learning model embedded features to the actual applications and objectives through online learning. In this framework, the latent factor model does not need to be updated very frequently or including the new items to make a good recommendation.

Besides the collaborative filtering methods, other factorization machine and embedding algorithms can be utilized in the same way [13, 32, 35]. Recent studies have shown many advantages of using the embedding algorithms to understand the complex data and provide useful features for search and ranking [20]. In comparison, many recently proposed algorithms use deep and wide neural networks to transfer high-variety input data to condensed features for the last layer to predict rewards. Moreover, our recommender system utilized segmentation and clustering approaches to generate interpretable features [34].

5 RESULTS

In this section, we will first describe couple simulation studies to compare the proposed naive linear contextual MAB algorithms with conventional MAB algorithms. These simulations are designed to test whether the additional assumptions and modifications will degraded performance under several common scenarios. In the second part of the results section, we will discuss two real-world usage examples of the recommender system in digital games. The purpose of showing real-world experiment results is to demonstrate that the recommender system can be easily configured to different applications and optimization objectives.

5.1 Simulations

For comparison of the MAB algorithms, we first built out empirical simulations with different size (d) of the contextual features and reward functions. To make the simulation easily repeatable by other researchers, we only use simple random variable generators to construct data. In the simulation, the probability of Bernoulli reward of each armed is computed by

$$r_a(x_t) = \frac{1}{\exp(-\eta_a^T x_t) + 1}, \tag{11}$$

where the reward coefficients vector $\eta_a \sim \mathcal{N}(0_{d+1}, \mathbf{I}_{d+1})$ is generated independently for each arm. The context feature vectors x_t are also generated from the standard Gaussian distribution $\mathcal{N}(0_d, \mathbf{I_d})$ with unit interceptor at each trial t. The average reward from 1000 repeats at each trial t is measured as a metric for comparison.

We also studied the situation where the variables in x_t do not span the entire d-dimensional linear space, in other words, highly correlated with each other. In the second part of the simulation, the x_t is generated with a random linear transform (generated from Gaussian distribution too) from the random vector with 100%, 90%, and 80% of d dimensions. Thus, the rank of feature variables is 100%, 90%, and 80% of the full rank d. Note that, since the linear transform are added to the feature generation process, the 100% rank with the linear transform simulation is expected to be different from the full rank case without the linear transform.

The simulation experiments include the following MAB algorithms:

1. Generic ϵ-greed: it randomly selects an arm with probability ϵ and selects the arm of the largest predicted reward with probability 1- ϵ, where ϵ is a predefined parameter.
2. Thompson Sampling (TS): it randomly draws the coefficients from the posterior distribution, and selects the arm of the largest predicted reward. The prior distribution is **Beta**$(0.5, 0.5)$.
3. Linear/Thompson Sampling: it makes use of the contextual features in a single linear model to estimate the posterior and selects the arm of the largest sampled reward.

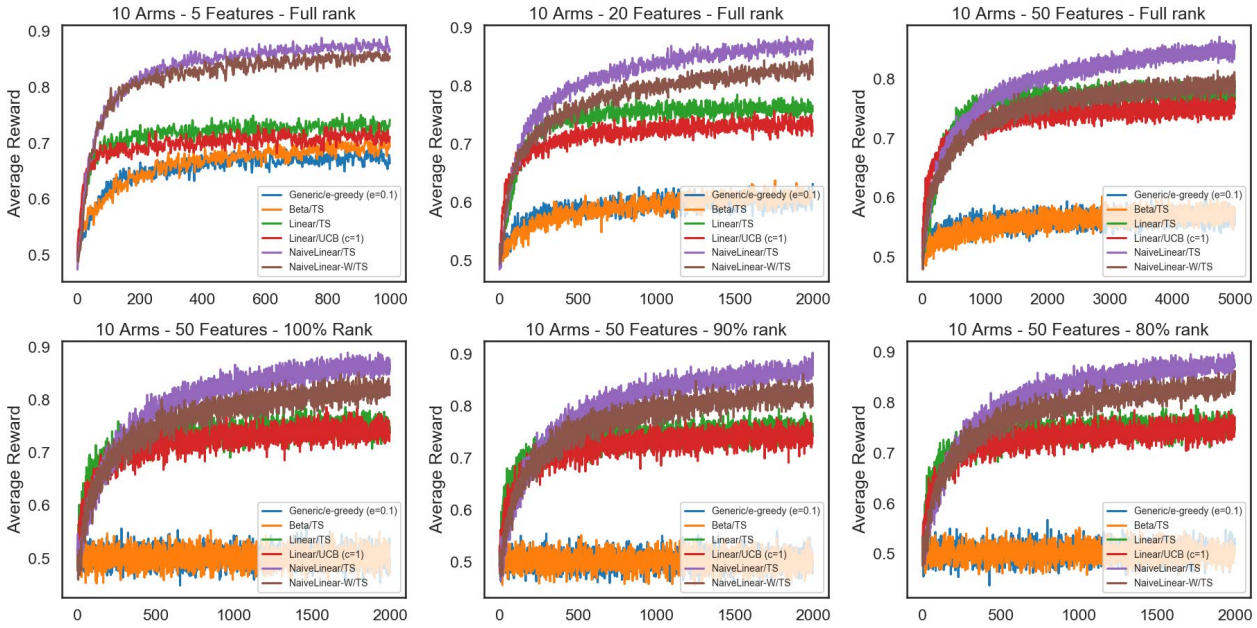

Figure 3: Simulation results to compare different MAB algorithm with different numbers of features and feature ranks.

4. Linear/UCB: it makes use of the contextual features in a single linear model to estimate the posterior and selects the arm of the upper confidence bound.
5. NaiveLinear/Thompson Sampling: it applies algorithm 3 with simple average to combine the estimated means.
6. NaiveLinear/Weighted/Thompson Sampling: it applies algorithm 3 with inverse of the prediction standard deviation as weighting to combine the estimated means.

The first part of simulations used ten arms, with 5, 20, 50 full rank independent feature variables in the contextual models. The average rewards are in the first row of Figure 3. When the number of the features is small, the naive linear models performed much better than the Linear Thompson Sampling or Linear UCB. Because the features are generated independently, the Naive Linear model correctly ignores the sample noise between the variables. From the 50 features simulation result at the top right, the differences between the linear model and naive linear model became smaller. We suspect that 5000 trials for 50 features model are not enough to fully converge. Also, the nonlinearly in Eqn. (11) introduces more inter variables relationship when the number of the feature variables increases.

The second row of Figure 3 shows the simulation results using the contextual features with 100%, 90%, and 80% of 50 dimensions. The naive linear model shows clear advantages in dealing with the ill-posed linear inversion problems over the original single linear model, although the empirical noise filled up the missing space.

In the simulations, the weighted version of the naive linear contextual MAB models underperformed the simple average version, because there was no data missing introduced to the simulations, and the variables were sampled from the identical distribution. Both of the scenarios were ideal for the average version. The advantage of the weighted version appears in the real-world data missing scenarios, which are not often to handle dynamically by the conventional linear models.

5.2 Real-world Experiments

Over the last two years, our recommender system has been integrated with over ten games on multiple platforms, and supported many use-cases. Because of the efficiencies gained by generalizations and centralized model, development and operations has required only a small group of data scientists and engineers. The end-to-end development and testing time for some simple applications were less than a week and achieved significant improvement over the baseline. All of contextual MAB models are updated automatically 24/7 without any routine manual works. Multiple experiments were conducted on almost every application for algorithm tuning and feature updates. In this section, we present two distinct applications: the first one is to recommend a new game to play in our game subscription service, and the second application is to recommend the game mode and map in a popular multiplayer game. The experiment results are evaluated at an aggregate basis.

In the first application, players who purchased the subscription service are free to download and play any games within the platform. The goal of the recommendation is to find the next favorite game for a player to enjoy. There are more than 100 games available in the subscription service. Game entitlement data with a board coverage were used for analysis and training the collaborative filtering model as well as the contextual MAB models. Figure 4 shows the new game installation rate of each experiment group. In this experiment, we compared the CF method and contextual MAB algorithm versus the manually curated orders. The contextual MAB method achieved up to 45% increases in installation compared to the control

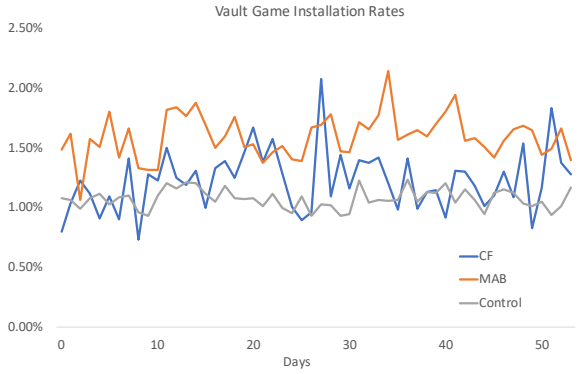

Figure 4: Experiment results for subscription game recommendation.

group, and the CF method also had about 10% improvement. The CF model had difficulties in capturing the up-to-date popularities and trends from the historical data. The hybrid model with CF latent features and MAB show a small further increase. In this application, we observed several significant shifts of interests in games caused by the new content released or other seasonality effects.

In the second application, the recommender system suggests a combination of map and game mode (such as a single player, cooperation, or competition) inside the game. Each time a player enters the main menu will see the recommendation from about 15 game modes and 20 maps. The goal is to find the most suitable match and potentially increase engagement. In this application, features used in the contextual MAB methods include gameplay durations, scores, levels, and skills for the current and previous games in the same franchise[3].

The experiment tested two MAB algorithms that were configured with two different optimization objectives: one is to optimize the click-through-rate (CTR), and the other one is to optimize the second-day retention rate. Figure 5 show the CTR of the game mode and map recommendation (top) and relative daily active users (DAU) changes over the control group (bottom). In the CTR measurement, both of the MAB methods increase the CTR. The direct CTR optimization group has about twice improvement over the group optimizing for the retention rate. The increase of CTR also raised for both of the MAB groups as the experiment continued running because more players preferred to use recommended quick join than other ways. In the bottom plot, we observed about 1% daily active users (directly link to the retention rate) increases by the MAB model configured to optimize for the retention, although its CTR improvement was not the best. It is interesting to observe the optimizing for short-term (CTR), and long-term (retention) have different effects.

6 CONCLUSIONS

We have presented a recommendation system that is designed for new opportunities and challenges in the digital game industry. The

[3]The recommendation does not affect the matchmaking or any game settings. Players can dismiss the recommendation to join any other wanted maps and game modes.

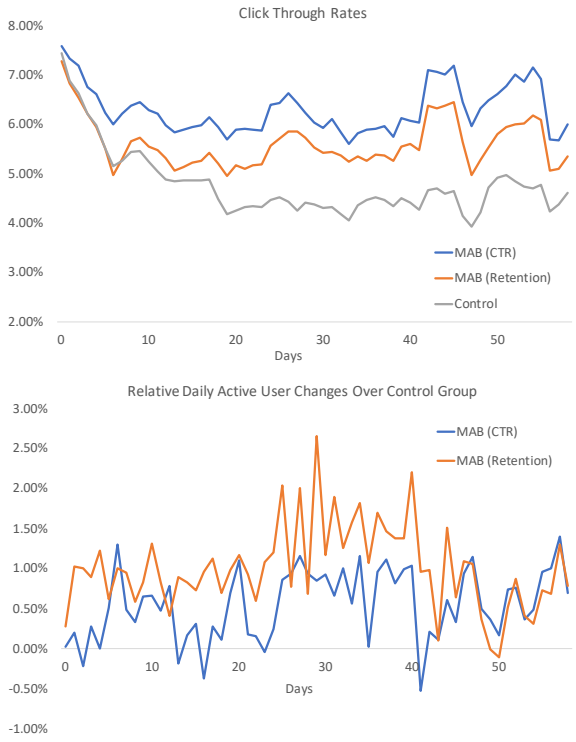

Figure 5: Experiment results of game mode and map recommendation.

recommendations offer suggestions to players, not forced choices, and also always honor relevant legal requirements, user account controls, and privacy settings. We have adopted two key strategies in the recommender system design. First, we prefer *systems over processes, and simplicity over complexity*. A system that enables simple game integration, model training, and execution is far more attractive than a highly sophisticated model that is difficult to train or apply. Second, we value *impact over accuracy*. The suggested selection or ranking produced by the recommendation model must serve the needs of the players and impact their experiences positively in order to have value. Similar lifts of KPIs by the recommender system were observed in many applications. The design to overcome the challenges of the heterogeneous and time-sensitive environment shows positive impacts on new content releases and live events. Since the recommender system and data platform is a centralized service, it becomes possible to have coherent and coordinated experience optimization cross games. The recommender system will become smarter with more data and learnings from new games.

ACKNOWLEDGMENTS

The authors would like to thank John Kolen, Jerry Hong, Hao Zhang, Yue Yu, Keran Li, Bingjie Sun, Wei Wang, Goutham Garimella, Harold Chaput, and Zebin Chen for their contributions.

REFERENCES

[1] Ernest Adams. 2010. *Fundamentals of Game Design*. Pearson Education. https://books.google.com/books?id=-BCrex2U1XMC

[2] Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* 17, 6 (2005), 734–749. https://doi.org/10.1109/TKDE.2005.99 arXiv:3

[3] Charu C. Aggarwal. 2016. *Recommender Systems*. Springer International Publishing, Cham. 29–71 pages. https://doi.org/10.1007/978-3-319-29659-3 arXiv:arXiv:1202.1112v1

[4] Shipra Agrawal and Navin Goyal. 2012. Analysis of Thompson Sampling for the multi-armed bandit problem. *JMLR Workshop and Conference Proceedings (COLT2012)* 23 (2012), 39.1–39.26. https://doi.org/arXiv:1111.1797 arXiv:1111.1797

[5] Shipra Agrawal and Navin Goyal. 2013. Thompson Sampling for Contextual Bandits with Linear Payoffs. In *Proceedings of the 30 th International Conference on Machine Learning*, Vol. 28. JMLR, Atlanta. arXiv:1209.3352 http://arxiv.org/abs/1209.3352

[6] Aliyun. 2017. Aliyun Recommendation Engine. https://data.aliyun.com/product/re

[7] Xavier Amatriain. 2012. Building Industrial-scale Real-world Recommender Systems. In *Proceedings of the Sixth ACM Conference on Recommender Systems, RecSys'12*. ACM, New York, New York, USA, 7–8. https://xamat.github.io/pubs/recsys12-tutorial.pdf

[8] Amazon. 2019. Amazon Personalize - Real-time personalization and recommendation, based on the same technology used at Amazon.com. https://doi.org/10.1016/j.ejogrb.2011.02.008

[9] Syed Muhammad Anwar, Talha Shahzad, Zunaira Sattar, Rahma Khan, and Muhammad Majid. 2017. A game recommender system using collaborative filtering (GAMBIT). In *2017 14th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. IEEE, 328–332. https://doi.org/10.1109/IBCAST.2017.7868073

[10] Peter Auer. 2002. Using Confidence Bounds for Exploitation-Exploration Tradeoffs. *Journal of Machine Learning Research* 3 (2002), 397–422. https://doi.org/10.4271/610369

[11] Robert M. Bell and Yehuda Koren. 2007. Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, Vol. 58. IEEE, 43–52. https://doi.org/10.1109/ICDM.2007.90

[12] Olivier Chapelle and Lihong Li. 2011. An Empirical Evaluation of Thompson Sampling. In *Advances in Neural Information Processing Systems 24*, . Shawe-Taylor Weinberger, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. (Eds.). Curran Associates, Inc., 2249—-2257. https://doi.org/10.1016/j.epsl.2005.07.019

[13] Heng-tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. Cheng2016, 7–10. https://doi.org/2988450.2988454

[14] Anthony Chow, Min Hui Nicole, and Giuseppe Manai. 2014. HybridRank : A hybrid content-based approach to mobile game recommendations. *CEUR Workshop Proceedings* 1245 (2014), 10–12.

[15] Evangelia Christakopoulou and George Karypis. 2016. Local Item-Item Models For Top-N Recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems - RecSys '16*. ACM, New York, NY, USA, 67–74. https://doi.org/10.1145/2959100.2959185

[16] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems - RecSys '16*, ACM (Ed.). 191–198. https://doi.org/10.1145/2959100.2959190

[17] James Davidson, Blake Livingston, Dasarathi Sampath, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, and Mike Lambert. 2010. The YouTube video recommendation system. *Proceedings of the fourth ACM conference on Recommender systems - RecSys '10* (2010), 293. https://doi.org/10.1145/1864708.1864770

[18] Anders Drachen, James Green, Chester Gray, Elie Harik, Patty Lu, Rafet Sifa, and Diego Klabjan. 2016. Guns and guardians: Comparative cluster analysis and behavioral profiling in destiny. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 1–8. https://doi.org/10.1109/CIG.2016.7860423

[19] Anders Drachen, Nicholas Ross, Julian Runge, and Rafet Sifa. 2016. Stylized facts for mobile game analytics. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 1–8. https://doi.org/10.1109/CIG.2016.7860392

[20] Mihajlo Grbovic and Haibin Cheng. 2018. Real-time Personalization using Embeddings for Search Ranking at Airbnb. In *Proceedings of The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Association for Computing Machinery, London, United Kingdom, 311–320. https://doi.org/10.1145/3219819.3219885

[21] Frédéric Guillou, Romaric Gaudel, and Philippe Preux. 2015. Collaborative Filtering as a Multi-Armed Bandit. In *NIPS'15 Workshop: Machine Learning for eCommerce*. Montréal.

[22] Negar Hariri, Bamshad Mobasher, and Robin Burke. 2014. Context adaptation in interactive recommender systems. In *RecSys 2014 - Proceedings of the 8th ACM Conference on Recommender Systems*. 41–48. https://doi.org/10.1145/2645710.2645753

[23] Volodymyr Kuleshov and D Precup. 2010. Algorithms for the multi-armed bandit problem. *Journal of Machine Learning* 1 (2010), 1–32. https://doi.org/10.1145/1109557.1109659 arXiv:arXiv:1402.6028v1

[24] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web - WWW '10*. ACM Press, New York, New York, USA, 661. https://doi.org/10.1145/1772690.1772758 arXiv:1003.0146

[25] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing* 7, 1 (2003), 76–80. https://doi.org/10.1109/MIC.2003.1167344 arXiv:69

[26] Michael Meidl, Steven Lytinen, and Kevin Raison. 2014. Using Game Reviews to Recommend Games. In *Proceedings of the Third Workshop on Games and NLP*. 24–29.

[27] Charles O Nutter, Thomas Enebo, Nick Sieger, Ola Bini, and Ian Dees. 2011. *Using JRuby: Bringing Ruby to Java* (1st ed.). Pragmatic Bookshelf.

[28] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. *Introduction to Recommender Systems Handbook*. Springer US, Boston, MA, 1–35. https://doi.org/10.1007/978-0-387-85820-3_1

[29] James Owen Ryan, Eric Kaltman, Timothy Hong, Michael Mateas, and Noah Wardrip-Fruin. 2015. People Tend to Like Related Games. *Proceedings of the 10th International Conference on the Foundations of Digital Games* Fdg (2015). https://games.soe.ucsc.edu/sites/default/files/ryanEtAl{_}BottomUpGameStudiesUsingNLP.pdf

[30] Rafet Sifa, Christian Bauckhage, and Anders Drachen. 2014. Archetypal game recommender systems. *CEUR Workshop Proceedings* 1226, January (2014), 45–56.

[31] Brent Smith and Greg Linden. 2017. Two Decades of Recommender Systems at Amazon.com. *IEEE Internet Computing* 21, 3 (may 2017), 12–18. https://doi.org/10.1109/MIC.2017.72

[32] George Trigeorgis, Konstantinos Bousmalis, Stefanos Zafeiriou, and Bjorn W. Schuller. 2017. A Deep Matrix Factorization Method for Learning Attribute Representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39, 3 (mar 2017), 417–429. https://doi.org/10.1109/TPAMI.2016.2554555 arXiv:1509.03248v1

[33] Qing Wang, Chunqiu Zeng, Wubai Zhou, Tao Li, S. Sitharama Iyengar, Larisa Shwartz, and Genady Grabarnik. 2018. Online Interactive Collaborative Filtering Using Multi-Armed Bandit with Dependent Arms. *IEEE Transactions on Knowledge and Data Engineering* (2018). https://doi.org/10.1109/TKDE.2018.2866041 arXiv:1708.03058

[34] Junyuan Xie, Ross Girshick, R B G F B Com, and Ali Farhadi. 2016. Unsupervised Deep Embedding for Clustering Analysis. In *Proceedings of the 33 rd International Conference on Machine Learning*, Vol. 48. JMLR, New York, New York, USA. arXiv:arXiv:1511.06335v2 https://arxiv.org/pdf/1511.06335.pdf

[35] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. 2017. Deep Matrix Factorization Models for Recommender Systems. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, California, 3203–3209. https://doi.org/10.24963/ijcai.2017/447

[36] Xiwang Yang, Yang Guo, Yong Liu, and Harald Steck. 2014. A survey of collaborative filtering based social recommender systems. *Computer Communications* 41 (2014), 1–10. https://doi.org/10.1016/j.comcom.2013.06.009

[37] Chunqiu Zeng, Qing Wang, Shekoofeh Mokhtari, and Tao Li. 2016. Online Context-Aware Recommendation with Time Varying Multi-Armed Bandit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*. ACM Press, New York, New York, USA, 2025–2034. https://doi.org/10.1145/2939672.2939878

[38] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep Learning Based Recommender System: A Survey and New Perspectives. *ACM Comput. Surv.* 52, 1 (feb 2019), 5:1—-5:38. https://doi.org/10.1145/3285029

[39] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A Deep Reinforcement Learning Framework for News Recommendation. In *Proceedings of the 2018 World Wide Web Conference (WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 167–176. https://doi.org/10.1145/3178876.3185994