Reinforcement learning algorithms for video game recommendations: a comparative study

Tomás Tapia*
Ignacio Villagrán*
tetapia@uc.cl
invillagran@uc.cl
Pontificia Universidad Católica de Chile
Santiago, Chile

ABSTRACT

Reinforcement Learning has proven to be a useful method in dynamic contexts, where the content is updated continuously. However, there are limitations for testing reinforcement learning techniques in recommender systems associated with receiving constant feedback from users interacting with a system (online testing). The purpose of this study is to evaluate the performance of recommender policies obtained through a reinforcement learning process in the context of video games' recommendation. The RecSim simulator [9] was used in a STEAM video games dataset. This simulator allows the generation of user behavior simulation based on configurable representations of the users and the content with which they interact. We present an environment that consists of a game and a user model. Using these two models, we defined an interaction model and a reward function used during the agent's training. For this work, on-policy (SARSA) and off-policy (q-learning) algorithms were compared. Each of these algorithms is run with two variants on the slate optimization process: top-k and greedy. The study's results allow us to conclude that, although SARSA presents a learning process when training with a single user, more training time is required for the algorithms to show their real potential.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computing methodologies** → *Reinforcement learning*; *Simulation environments.*

KEYWORDS

reinforcement learning, recommender systems, sequential decision making

1 INTRODUCTION

The videogame market currently stands out as one of the most profitable and promising businesses [14]. This industry has managed to place itself as one of the most valued worldwide, growing substantially in sales and users during the pandemic. In particular, STEAM, one of the most important platforms in the world of video games, achieved this year the impressive record of 20 million users connected simultaneously, with 6.2 million of them interacting with some game [1]. Thus, the number of games available increases at high speed and the amount of information regarding their genres and categories is getting more and more significant. This difficult the task of selecting a new game for the user [5]. Because of this, Reinforcement Learning (RL) has emerged as a methodological alternative to traditional recommending techniques that has proven to be useful for recommendations hosted in dynamic and constantly changing environments.

Numerous advantages in the use of RL have been described. For this project, we concentrate on two. First, the use of RL has been reported in constantly dynamic contexts. Secondly, methods are needed to counteract the fact that recommendation systems consistently deliver similar content where it is possible to add a percentage of randomization to recommendations through exploration strategies [17]. In the context of video games, [5] presents matrix factorization techniques and sentiment analysis of user reviews on the STEAM dataset [10, 13, 16]. However, specifically in RL, limited evidence has been found concerning that applied to this dataset. Given the changing nature and increasing incorporation of new video games into this platform, we believe that using RL as a strategy to optimize recommendations could benefit both users looking for new games and small game creators looking to be recommended.

This project's overall objective is to evaluate the performance of recommendation algorithms obtained through a reinforced learning process in the context of video game recommendation.

2 RELATED WORK

Many websites need to quickly adapt their services to deliver customized content to their users in a context where new content is continually introduced into the system. Thus, RL has been reported in changing contexts, such as news recommendation, where contents are quickly outdated, and user preferences can evolve, making traditional methods, like collaborative filtering, inapplicable. In this regard, Li et al. address the recommendation of news articles as a problem of multi-armed bandits, which seeks to select the next item to recommend considering the potential items as arms of a bandit. The authors proposed LinUCB, an algorithm that considers contextual information about users and items intending to maximize total user interactions by evaluating them directly from logged events, thus avoiding problems associated with online evaluation and evaluation based on simulated information [12].

In a music recommendation environment, Bendada et al. model a carousel personalization as a contextual multi-armed bandit problem, demonstrating effectiveness through the representation of

*Both authors contributed equally to this research.

user behaviors against music playlists and semi-personalization through user grouping [3]. Lacerda et al. provide another example of dynamic systems placed in a Daily-Deal Sites context. This scenario is incredibly challenging since past user preferences are very sparse, deals have a very short and changing lifespan, and customer preferences change rapidly over time. Because of this, the authors proposed a new algorithm based on the explore-exploit strategy. In this algorithm, the recommendation structure is updated based on feedback from a part of the users in the exploration phase. On the other hand, this updated network is used to improve the recommendation algorithm for the rest of the clients in the exploitation phase, demonstrating overall better performance metrics depending on the user splitting strategies [11].

The explore-exploit approach for multi-armed bandits is extracted from sequential recommendations, especially under uncertainty. It includes exploring through the recommendation items that provide more information about the users and exploiting them when obtaining immediate feedback of the highest possible quality. Studies indicate that this approach demonstrates better accuracy of personalized recommendations in short times, highlighting their real applicability. In this environment, ϵ-greedy is distinguished as a simple but efficient approach to balance exploration and exploitation by selecting the best value arm, greedy action (exploits), and selecting a random action (explores) [2, 7].

In the field of adaptive personalization and item diversification, Teo et al. experimented with the Amazon fashion website, seeking to improve items' visual recommendation. In their article, the authors describe an algorithm that presents the user with a diverse set of exciting elements while adapting to the user's interactions. This objective is achieved through a Bayesian regression model that scores item relevance. This model reorders the top-scoring items based on category and customized preferences learned from each user's behavior. The proposed solution resulted in a substantial increase in the number and time of interactions [15].

Ultimately, the literature has described that well-configured sets of recommending systems can achieve greater efficiency than separate combined systems. In this regard, Cañamares et al. explored the adaptation of a multi-armed bandit approach in an interactive context with the aim that the set observes and learns about the combined algorithms' effectiveness and improves its configuration progressively. This study managed to demonstrate effectiveness and superiority in interactive environments [4].

3 PROPOSED SOLUTION

The existing limitations for testing RL techniques in recommender systems are associated with the need to receive constant feedback from users interacting with a system (online testing). To avoid this problem, we proposed to use the RecSim simulator. This simulator allows the generation of user behavior simulations based on configurable representations of the users and the content with which they interact. This simulator was created for the study of recommendation systems based on RL techniques [9]. Based on the above, we carry out experiments with both on-policy and off-policy algorithms.

RecSim proposes a way of developing an environment which consists of creating a User Model, a Document Model and an Interaction (or Response) Model. At each timestep, the simulator samples a user and a number of candidate documents, and feeds this information to an Agent. The agent then generates a k-sized slate, which is a set of k documents recommended to the sampled user. The user, finally, receives this slate and chooses one or more items to interact with, revealing some information to the agent, and also updating its preferences, given the new document it has seen. This loop can be visualized in Figure 1.

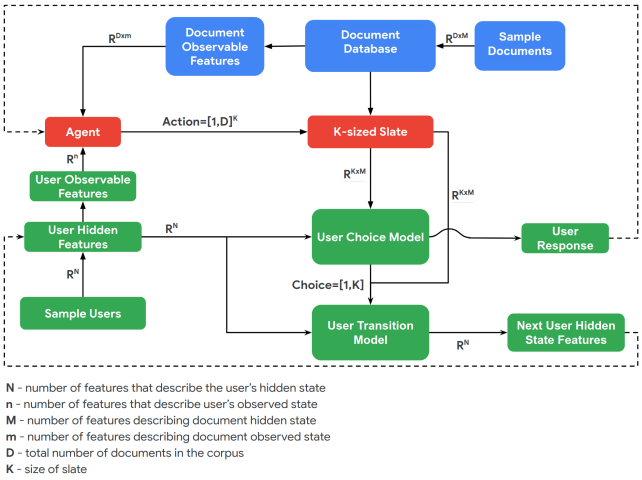

Figure 1: Data Flow through RecSim components. (Ie et al. [9])

Using the previously described tool, we develop a new environment based on the STEAM dataset, generating users, documents and interaction models. Then, we run some experiments using agents provided by RecSim. We compare four methods proposed by Ie et al based on the assumptions presented in their work. The authors of that paper suggest that the value of a slate of items can be decomposed into a function of its components, i.e. the value of an item does not depend on the slate on which it is presented [8]. Thus, we run experiments using an on-policy algorithm (SARSA) and an off-policy algorithm (Q-Learning). Each of these algorithms is run with two variants on the slate selection process: top-k and greedy, which are explained in detail in section.

4 DATASET

Concerning the data, we propose to use the STEAM Video Game and Bundle Data [10, 13, 16], whose general information is shown in Table 1.

First, the database was filtered as proposed in [10], eliminating items and users with less than five interactions. Table 2 describes the number of final users and items and the descriptions of each one's interactions. The dataset' sparsity was 0.99, mainly due to the large number of users using the platform.

Regarding users, we note that the average is in 12 interactions, and analyzing the data we observe a non-normal long tail distribution. The items follow a similar distribution to the users, with the

Table 1: STEAM dataset variables

Variables	n
Reviews	7.793.069
Users	2.567.538
Items	32.133
Bundles	615

data grouped in smaller values and with a long tail distribution. This allows us to conclude that there are a small number of very popular items within a large universe of games with few interactions.

If we take the five users with the most interactions, we can see that they represent a low percentage of the total interactions. On the other hand the items with more interactions represent about 1% of the total interactions, which is somewhat relevant given the large number of total interactions available.

Finally, the items were grouped according to the tags and specifications that were present in the database. The items contain, in total, 365 tags and the 20 most used tags were selected for visualization, which are shown in Figure 2.

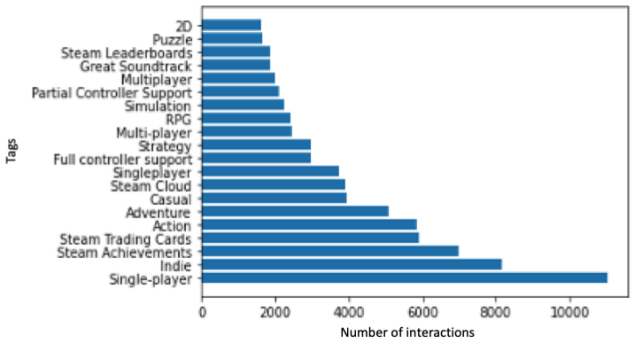

Figure 2: Number of interactions of the 20 most used tags.

5 METHODS

5.1 Steam Dataset Environment

We present an environment that consists of a document model (games) and a user model. Using these two models, we defined an interaction model and a reward function used during the agents training.

Firstly, we generate a document model that stores an ID associated with a specific game and the information from the tags associated to that game. In other words, a game is represented by the labels it presents in the Steam dataset. Based on this, we utilize the 100 most common labels (out of the available 365) to describe the game through a vector containing a one if the game has the label and a zero if it does not. A way to incorporate additional information about the game in this model, such as price, developer, or review value, is being evaluated. These variables could be incorporated by normalizing their values, comparing them with all the games in the database, and finally integrating them as an additional value in the tag characteristics vector.

Secondly, we define a user model that determines its hidden and visible states. We determined that a user has a vector that describes his affinity with each of the existing database labels for the visible state. In this way, the vector associated with the user shares the same dimensionality as the vector associated with a game. For the user's hidden states, we determine that each user has latent factors that represent the users' actual preferences, which, when multiplied by an item latent factors, results in the expected user value.

Taking both models into consideration, we develop a response (interaction) model, which determines how each user interacts with a slate of presented items. We score every item in the slate of items presented to a user by computing the Manhattan distance between the vector representing the visible state of the user (label preferences) and the vector that represents the state of the document. These scores are then fed into a Multinomial Logit Choice Model, which generates a probability for each item on the slate. Each of these probabilities represents the chance that a user has to buy that item in particular (and not buy the rest). For the sake of simplicity, a user will always purchase only one of the recommended items.

We also determine how a user state evolves when interacting with a new item. Thus, we determine that every time a user purchases a new game, its state will become an average of the previous state and the item state. By defining this user transition, we give the latest interactions a higher weight in determining the user state than those earlier in the past.

Having defined every element of our environment, we are left only with the definition of a reward function, which will guide the different agents' learning process. We take advantage of the hidden state defined in the user model and determine that the agent will have access to the actual value that this user gives to that item every time a user purchases an item. Thus, we define the reward function (generally) as follows:

$$R(u, S) = \sum_{i \in S} c_{u,i} \cdot p_i^\top q_u$$

The reward associated with a user u and a state S corresponds to the sum of the actual values of the slate's purchased items. In the previous definition, $c_{u,i}$ becomes 1 when the user purchased item i and 0 otherwise; p_i and q_u represent the item and user latent factors. Given that every iteration only a single item is purchased, the equation above becomes the purchased item's value.

5.2 Initial Parameters

For each user in our preprocessed dataset, we compute its visible state by determining its preference over the different tags present in the user model. Each preference value vector is defined as follows:

$$pref(u, t) = \frac{1}{|P_u|} \sum_{i \in P_u} b_{i,t}$$

In the previous definition, P_u is the set of items purchased by the user u in the original dataset, and $b_{i,t}$ is the value that the t label has associated with the item i, which can be 1 if it is present or 0 if it is not. Consequently, the value of user preference u for the label t is an average of the values of the labels of the items that the user u has purchased.

Table 2: Descriptive analysis of the filtered STEAM data set

Variables	n	Interactions				
		Mean	Std	Median	Min	Max
Revisions	3.484.694					
Unique users	281.210	12,39	19,62	8,0	5	1.226
Unique items	11.961	291,34	993,15	40,0	5	31.699

We then utilize a matrix factorization method (ALS) to generate a ground truth to be used as the users' hidden state. Thus, we run the model mentioned above in our training data and generate user and item factors, which are stored in the user and document models.

When sampling a user, the simulator samples a random pair of hidden and visible states from the generated data associated with a single user. Similarly, when sampling a document, the simulator samples a random pair of labels and item factors associated with a single item.

5.3 Agents

We run experiments using an on-policy agent (SARSA) and an off-policy agent (Q-Learning). SARSA is a method that incorporates and utilizes the knowledge it acquires during the training process. It is on-policy because it uses the same policy it is learning from the data it receives. Q-Learning is a method that uses the knowledge it acquires to learn, but it does not use it regularly during the training process. It is off-policy because the policy it uses to recommend during the training phase is not the same as learning "behind the scenes".

Each of these algorithms is run with two variants on the slate selection process: top-k and greedy. Top-k selects k items with their highest unnormalized expected long-term values, while greedy selects items and updates their scores with relation to the current partial slate. In the first case, the algorithm only considers the expected value to the user. In contrast, the second one considers the probability of examination, which corresponds to the user's score to the items a priori (in our case, a similarity metric between the user-visible state and the item state).

As a baseline method, to compare whether or not the algorithms we are testing are learning to recommend, we utilize a Random selection of the slates. This algorithm does not utilize any information provided by the environment, selecting every slate randomly from the available candidates.

5.4 Training Setup

We run each of the algorithms for 100 iterations, and in every iteration a user responds to a recommended slate a total of 200 times. Thus, the length of training adds up to 20k timesteps. This means that we simulate 100 different users and their interactions with 200 sequentially recommended slates. Due to memory limitations, at each iteration we sample 1% of the items at random. This means that during a single iteration only a limited number of videogames are available for recommendation. These limitations are further explained in section 7. The target network update frequency for the Q-Learning algorithm was set at 8k steps. Thus, this update was executed two times during the full training period.

6 RESULTS

Figure 3 represents the training of the algorithms in this environment. We observe a significant variability in the algorithms' behavior. No algorithm manages to improve substantially, only highlighting the q-topk as it begins to rise slightly but without a clear trend. This result can be given because as the system has to see many users, it probably did not learn. After all, it could not see the users more than once.

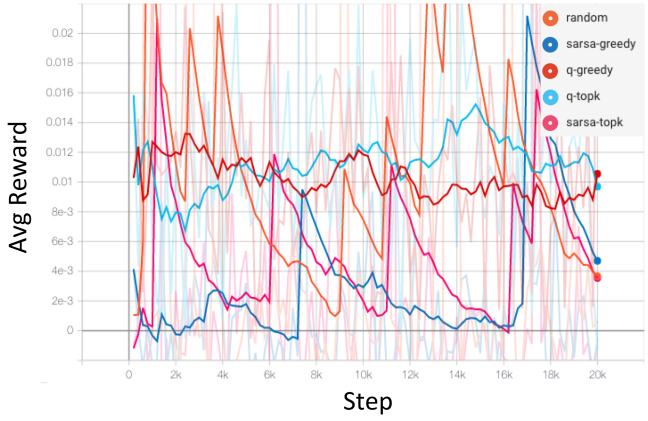

Figure 3: Average rewards per episode for regular environment (Smoothing = 0.9).

Consequently, we asked ourselves if the agents could have a better performance if interacting with a single user for the same amount of time. Thus, we tested with only one user (Figure 4). In this case, the user had the most interactions in the dataset (since we had more information). We observed that SARSA has better performance, probably because it applies the policy that it is learning instantly. On the other hand, q-learning performs more like a random algorithm, as it could not apply the policy it was learning and would require more time to explore the recommendation policy better sufficiently. It should be noted that only q-learning greedy and SARSA greedy were used since in previous analyses, the results between top-l and greedy were similar concerning the slate selection policy and, given this, the comparison between off and on-policy was prioritized.

It is worth noting that the number of training steps were relatively low due to a slow performance limitation, which is better explained in section 7. However, these low time steps are relevant when trying to explain the extremely poor results obtained when using the Q-Learning algorithms. These algorithms rely on a periodic update of the neural network that estimates the Q-value for

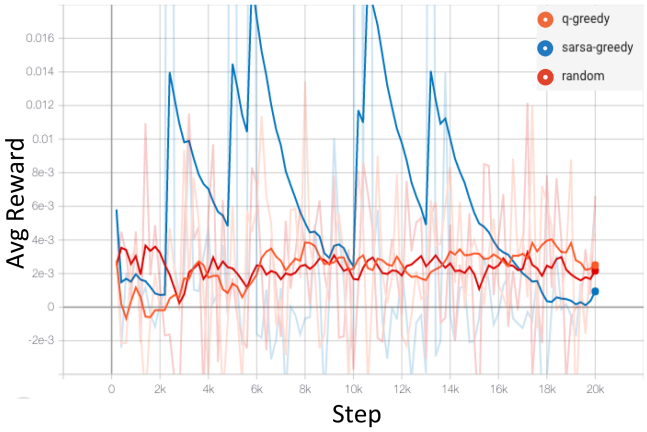

Figure 4: Average rewards for a single user environment (Smoothing = 0.9).

the items. We infer that the number of updates of this network was not enough to generate an increase in performance.

7 CONCLUSIONS AND FUTURE WORK

Based on the above results, we can conclude that there is a learning process when training in a single-user environment (specifically using SARSA), but there are relevant limitations when extending this to more users. The Q-learning algorithm does not complete enough steps to learn and then optimize the recommendation policy for both one and overall user environments. This is why more research is required to further tune the parameters used in the training process.

On one hand, we observed that there is not enough evidence to support the relationship between the tag model and the real hidden relevancies. This is because we noticed that the tag model, which gives the user a probability of selecting an item, does not substantially increase the reward. We were only able to verify an approximation that it could increase the relationship in a single user; however, this is not representative because the user with the most interactions was selected. A step towards improving this selection model would be to incorporate the position of an item in the ranking, as has been thoroughly studied in dynamic learning-to-rank problems [6].

On the other hand, we believe that our approach to this problem would have achieved better results, if we had extended the training timesteps, at least in the single-user environment. This has a strong relation with the default parameters used by the original SlateQ model. Having such long time step intervals between the network updates partially explains the bad results.

There is room for improvement in almost every aspect of our work, but we believe that our initial motivation was incorrect, given the tools that we used. RecSim is a very powerful tool that allows for research that goes beyond achieving good accuracy results with a static dataset. A better (and earlier) understanding of RecSim's capabilities would have allowed us to improve the way we model users, video games, and the interactions that are simulated between them. A RecSim-esque way to model this problem would have been to define probability distributions over the hidden and observable features that describe the elements of the environment, and that could have been done by further analyzing and understanding the contextual information that is present in the dataset.

8 LIMITATIONS

The first limitation we can find was the way we used this dataset. By sampling the real values that represent the items, we faced a problem during the training process. The algorithms we tested try to find the Q values, representing the value that a particular user state will find by buying a specific item. However, the memory used by these algorithms is directly proportional to the number of items for which they try to find Q values. Consequently, we ran out of RAM during the training setup, both with the complete dataset and with 10% of it. This could be explained because we decided to sample a small portion of the dataset at every timestep. Additionally, it is worth noting that we worked with Google Colab free tier, which has approximately 12GB memory capacity.

Another limitation we found using RecSim was the minimal number of issues reported in their official repository because this tool was released very recently. So, we frequently faced errors for which there was no further information online. This led us to carefully inspect the repository code and other tools used by RecSim (such as Dopamine).

Finally, we had trouble dealing with the evaluation process, also due to insufficient RAM. We were unable to reload the models of SARSA and Q-Learning that we trained. This had a significant impact on our results because we could not generate a recommendation outside of the training split.

REFERENCES

[1] [n.d.]. Steam Charts. https://steamdb.info/graph/. Accessed: 2020-10-13.
[2] Andrea Barraza-Urbina and Dorota Glowacka. 2020. Introduction to Bandits in Recommender Systems. In *Fourteenth ACM Conference on Recommender Systems*. 748–750.
[3] Walid Bendada, Guillaume Salha, and Théo Bontempelli. 2020. Carousel Personalization in Music Streaming Apps with Contextual Bandits. *Fourteenth ACM Conference on Recommender Systems* (Sep 2020). https://doi.org/10.1145/3383313.3412217
[4] Rocío Cañamares, Marcos Redondo, and Pablo Castells. 2019. Multi-armed recommender system bandit ensembles. 432–436. https://doi.org/10.1145/3298689.3346984
[5] Germán Cheuque, Jose Antonio Guzman Gomez, and Denis Parra. 2019. Recommender Systems for Online Video Game Platforms: the Case of STEAM. 763–771. https://doi.org/10.1145/3308560.3316457
[6] Craswell, Nick, Zoeter, Onno, Taylor, Michael Lyu, Ramsey, and Bill. 2008. An experimental comparison of click position-bias models. *WSDM'08 - Proceedings of the 2008 International Conference on Web Search and Data Mining*. https://doi.org/10.1145/1341531.1341545
[7] Frédéric Guillou, Romaric Gaudel, and Philippe Preux. 2016. Scalable explore-exploit Collaborative Filtering. In *Pacific Asia Conference on Information Systems (PACIS'16)*. Chiayi, Taiwan. https://hal.inria.fr/hal-01406418
[8] Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Tushar Chandra, and Craig Boutilier. 2019. SlateQ: A Tractable Decomposition for Reinforcement Learning with Recommendation Sets. 2592–2599. https://doi.org/10.24963/ijcai.2019/360
[9] Eugene Ie, Chih wei Hsu, Martin Mladenov, Vihan Jain, Sanmit Narvekar, Jing Wang, Rui Wu, and Craig Boutilier. 2019. RecSim: A Configurable Simulation Platform for Recommender Systems. arXiv:1909.04847 [cs.LG]
[10] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. arXiv:1808.09781 [cs.IR]
[11] Anisio Lacerda, Rodrygo Santos, Adriano Veloso, and Nivio Ziviani. 2014. Improving daily deals recommendation using explore-then-exploit strategies. *Information Retrieval Journal* 18 (12 2014). https://doi.org/10.1007/s10791-014-9249-4
[12] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. *Proceedings of*

the 19th international conference on World wide web - WWW '10 (2010). https://doi.org/10.1145/1772690.1772758

[13] Apurva Pathak, Kshitiz Gupta, and Julian McAuley. 2017. Generating and Personalizing Bundle Recommendations on Steam. 1073–1076. https://doi.org/10.1145/3077136.3080724

[14] Jason Schreier. [n.d.]. Gaming Sales Are Up, but Production Is Down. *The New York Times* ([n. d.]). https://www.nytimes.com/2020/04/21/technology/personaltech/coronavirus-video-game-production.html

[15] Choon Hui Teo, Houssam Nassif, Daniel Hill, Sriram Srinivasan, Mitchell Goodman, Vijai Mohan, and S.V.N. Vishwanathan. 2016. Adaptive, Personalized Diversity for Visual Discovery. *Proceedings of the 10th ACM Conference on Recommender Systems* (Sep 2016). https://doi.org/10.1145/2959100.2959171

[16] Mengting Wan and Julian McAuley. 2018. Item recommendation on monotonic behavior chains. 86–94. https://doi.org/10.1145/3240323.3240369

[17] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A Deep Reinforcement Learning Framework for News Recommendation. *WWW '18: Proceedings of the 2018 World Wide Web Conference*, 167–176. https://doi.org/10.1145/3178876.3185994