Recommender System MDP with Reinforcement Learning

COMPSCI 687 Final Project Report Fall 2023

Aman Singh Thakur[1,*] and *Prachi* Jain[1,**]

[1]University of Massachusetts Amherst, USA

Abstract. Recommender systems are traditionally viewed as static entities primarily engaged in prediction tasks. Contrary to this perspective, the paper "An MDP Based Recommender System" [1] posits that the generation of recommendations should be regarded as a dynamic and sequential decision-making process. To substantiate this hypothesis, we adopt the framework of Markov Decision Processes (MDPs) for Recommender systems. This project involves the implementation of MDPs, followed by the deployment and evaluation of various reinforcement learning techniques.

*e-mail: amansinghtha@umass.edu

**e-mail: ppjain@umass.edu

Contents

1 Introduction

1.1 Recommender Systems

A Recommender System [2] is designed to furnish personalized suggestions or recommendations to users. The primary objective of these systems is to forecast the preferences or interests of users and present items or content deemed pertinent and enjoyable to the user.

Several popular recommender systems have achieved notable success across diverse domains. Netflix's [3] recommender system, for instance, excels at suggesting movies and TV shows by leveraging sophisticated algorithms that analyze a user's viewing history, preferences, and ratings. Amazon's [4] recommendation engine is instrumental in driving sales by intelligently suggesting products based on a user's purchase history and browsing behavior. YouTube's [5] recommender system enhances user engagement by recommending videos tailored to individual preferences, taking into account watch history and search queries. Spotify's [6] success lies in its ability to curate personalized music recommendations, considering a user's listening history, liked songs, and genre preferences. These recommender systems, along with others like LinkedIn [7], Google News [8], and Pandora [9], have effectively harnessed the power of algorithms to provide users with content, connections, or products that align closely with their interests and behaviors. The success of these systems can be attributed to their capacity to continuously learn and adapt, leveraging user data and behavioral patterns to refine and enhance recommendations over time. This adaptability, coupled with a focus on personalizing and user-specific experiences, has contributed significantly to the widespread acceptance and effectiveness of these recommender systems in their respective domains.

Recommender systems encompass various approaches to enhance accuracy and personalization. Collaborative filtering [10] analyzes user behavior, either by user or item, to provide recommendations. Content-based filtering [11] relies on item features and user preferences. Matrix factorization [12] decomposes interaction matrices to uncover latent factors for personalized suggestions. Hybrid recommender systems [13] integrate multiple methods for robust recommendations. Knowledge-based systems [14] leverage explicit information about items and users, while context-aware systems consider factors like time and location for more relevant suggestions. These diverse types of recommender systems cater to different data and user scenarios, contributing to the effectiveness and versatility of personalized recommendation engines.

2 Methodology

2.1 Dataset

The dataset, a synthetic collection curated from Kaggle[1], comprises three CSV files: "games.csv," "transactions.csv," and "user-id.csv." It offers a detailed simulation of user interactions within a gaming environment.

"Games.csv" catalogues a variety of games, presenting essential data like unique game IDs, titles, and prices. This file includes entries for popular titles such as Counter-Strike Condition Zero, Half-Life 2 Deathmatch, and Terraria, each accompanied by its game ID and price.

[1] https://www.kaggle.com/datasets/arnabchaki/popular-video-games-1980-2023

"Transactions.csv" tracks user transactions, detailing user IDs, game titles, types of behavior (such as purchase or play), and their respective values. Each record in this file signifies either a purchase or gameplay activity by a user for a specific game, along with quantifiable data.

The "user-id.csv" file compiles a list of unique user IDs, representing the diverse user base in the dataset. These IDs are indicative of individuals who have interacted with the games, either through buying them or engaging in gameplay.

Collectively, these files create a rich, synthetic dataset that mirrors real-world user behavior in gaming. Analysis of this data can uncover patterns in user preferences, game popularity, and overall engagement trends, providing valuable insights for understanding and optimizing user experience in digital gaming platforms.

2.2 MDP Formulation for Recommender Systems

We've taken our inspiration from paper 'An MDP-Based Recommender System' [1] where authors Shani et. al have discussed how their MDP-Based Recommender System helps them take into account long term effects of each recommendation and the expected value of each recommendation. Further, They are able to replicate the MDP's performance for an actual website.

- We have formulated our problem as a discrete state, discrete action MDP. By restricting movement along the vertical axis, we reduce the number of valid actions that the agent can perform.
- The major issue we faced while formulating the MDP was to identify a compact state representation and keepingthe unique state count in check. It can be difficult for a policy to learn given too many states and less examplesto learn from.

The key components of our Recommender Systems based MDP is as follows -

- **States (S):** $S = \{s_1, s_2, \ldots, s_n\}$

  In the context of analyzing the sequence of the last k purchases, a First Order Markov Chain (MC) is employed to construct the state space. The Markov chain comprises a set of states, a stochastic transition function representing the probability distribution over states at sequence point t, given the state at sequence point t-1, and an initial probability distribution over states. These sequences are denoted as vectors of size k, specifically represented as $s_i =< x_1, \ldots, x_k >$, signifying the state in which the user's last k selected items were $< x_1, \ldots, x_k >$.

  Note: Sequences with fewer than k items (denoted as $l < k$) are converted into vectors, wherein x_1 through x_{k-l} take on the value "missing." For instance, considering k=3, some of the states (in arbitrary order) include $< None, None, x_1 >$, $< None, x_1, x_2 >$, $< x_1, x_2, x_3 >$, $< x_2, x_3, x_4 >$, $< None, None, x_{20} >$, and $< x_5, x_{17}, x_{19} >$.

  The total number of states for k=3 can be determined by a specific formula, $|A|^S$.

- **Actions (A)**: $A = \{a_1, a_2, \ldots, a_n\}$ In this MDP, we transition from one state to another by performing action $a_i \in A$. The action space A comprising of n actions i.e n items $x_1, x_2, ..., x_n$.

  For example, Performing action x_i is equivalent of buying item x_i and transitioning to a state where x_i is part of the new state, if the user accepts the recommendation.

- **Transition Function and Probabilities (P):** $P(s' \mid s, a)$
  The transition function for our Markov chain describes the probability that a user whose

k recent selections were $s =< x_1; \ ::: \ ; x_k >$ will select the item x' next. This is denoted by $tr_{MDP}(s, x', s')$ where s is the current state and s' is the next state. If a agent transitions from a state s to s', we can calculate the prediction of that happening using $tr_{predict}(s'|s, x')$ which is defined as follows -

$$tr_{predict}(s'|s, x') = \frac{count(s, s')}{\sum_{s'} count(s, s') * k} \tag{1}$$

Since the user can accept the Recommender System recommendation and buy item x' or reject it and another buy item. To model this accurately, we've to make this MDP stochastic by introducing two hyper-parameter weights α and β to give weights to decision on buying the item that recommender recommended and buying a different item from the one recommender recommended.

α - A recommendation increases the probability that a user will buy an item. This probability is proportional to the probability that the user will buy this item in the absence of recommendations

β - The probability that a user will buy an item that was not recommended is lower than the probability that she will buy it in the absence of recommendations, but still proportional to it

The final transition probability of a given state would look like $tr_{MDP}(s, x', s')$.

$$tr_{MDP}(s, x', s') = \alpha * tr_{predict}(s'|s, x') + \beta * \sum_{s' \neq s''} tr_{predict}(s''|s, x') \tag{2}$$

- **Reward Function** (R): $R(s, a)$

  We have modified the reward function for our Markov Decision Process (MDP) to align with our dataset. The reward to transition to any state, denoted as $R(s, a)$, is now defined as the ratio of the total time spent playing all games in that state to the cumulative sum of the prices of all the games. In this context, the state s is represented as $s =< x_1, \ldots, x_k >$.

  The updated mathematical expression for the reward function is as follows:

$$R(s, a) = \frac{\text{Total Time Spent Playing all Games in State s and Game a}}{\text{Cumulative Sum of Prices of all Games in State s and Game a}}$$

  Here, the numerator represents the total time spent playing all games in the given state s, and the denominator is the cumulative sum of the prices of all games. This modification captures the ratio of the aggregated playtime to the economic value represented by the sum of game prices, providing a more customized representation for our specific dataset within the MDP framework.

- **Discount Factor** (γ): $0 \leq \gamma \leq 1$ Discount Rate γ calculates how myopic the agent is. For γ close to 0, agent will give preference to transitioning to states that give the best immediate reward while a γ close to 1 will allow the agent to give preference to future rewards that are more than the immediate rewards.

- **Deterministic Policy** π: In the context of a scientific investigation, consider a state represented as s = <x1, x2...xk> with a set of possible actions <x1, x2....., xn>. The user is confronted with the decision to either accept a recommendation x' and transition to the

subsequent state <x2...xk, x'>, or reject the recommendation, opting for an alternative action x" and transitioning to the state <x2, xk, x">. Both transitions occur with a certainty of 100%. Consequently, our policy π is designed as deterministic, wherein only one action is prescribed for a given state, defining an optimal policy.

The deterministic nature of the policy enables the application of reinforcement learning techniques such as Monte Carlo and TD Learning, which inherently rely on a deterministic policy. This quality facilitates the utilization of these techniques in our scenario.

- **No Terminal State:** Our MDP is infinite horizon because it captures the ongoing real world continuous essence
- **Initial State:** For each user, the initial state corresponds to their first transaction.

  For example, if the user purchases item x_1,its intial state will be $< None, None, x_1 >$
- **Example Dynamics Overview:** Figure 1 illustrates the dynamics inherent in the Markov Decision Process (MDP) when incorporating information from the user's preceding three purchase instances. Initiated at state $s =< x_1, x_2, 23 >$ and time t, the stochastic nature of the MDP enables transitions to various states with a transition probability denoted by tr_{MDP}. The model internally accounts for whether a given action was recommended by the system (*alpha* weight) or not (*beta* weight). In this instance, the user moves to state $s' =< x_2, x_3, ..., x_k >$ through the transition $tr_{MDP} < s, x_4, s' >$. At this point, the agent loses information about x_i or any transaction occurring prior to the kth purchase, allowing for subsequent transitions based on the stochasticity of the new state. This MDP ensures recommendation quality based on the last k purchases, and its stochastic nature facilitates both exploitation and exploration, resulting in optimal recommendations assessed by the Recommendation Score metric.

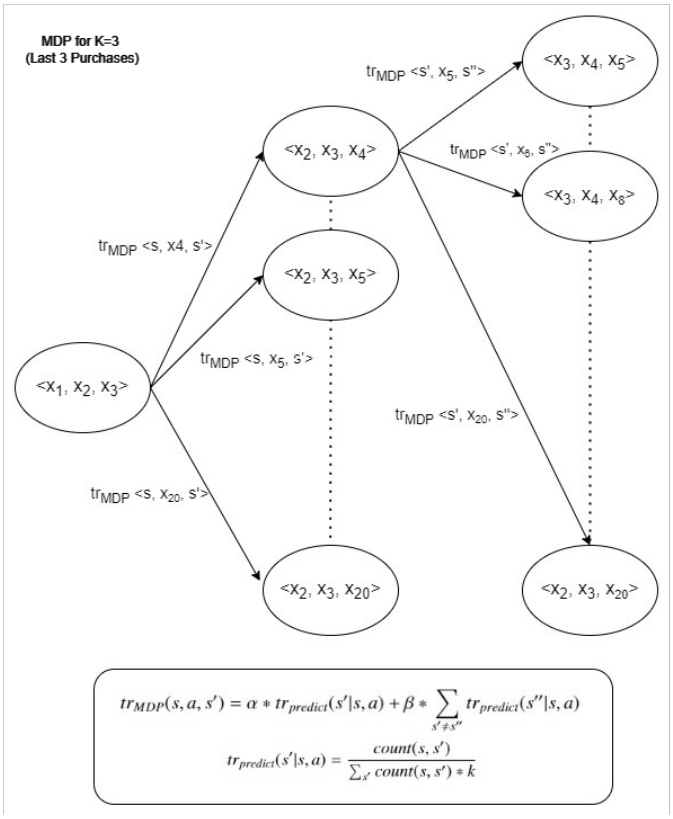


Figure 1: Recommender System MDP Transitions and Dynamics Example

2.3 Experiments

2.3.1 Randomized Algorithm for Optimal Policies

As discussed in Homework 1, a simple randomized algorithm for identifying optimal policies in a Markov Decision Process (MDP) context is extremely beneficial as it converges to a optimal policy in N iterations.

The key intuition behind this algorithm is to explores the policy space by randomly sampling policies. This can be particularly useful in complex environments where the policy space is too large to analyze exhaustively. While there's no guarantee of finding the absolute best policy, there's a probability of identifying highly effective policies, especially if the number of iterations (N) is large.

In a Recommender System MDP using Reinforcement Learning, the simplicity and exploratory nature of this randomized algorithm make it a useful tool, especially in complex environments like recommender systems. It can provide a quick baseline for policy performance and serve as a starting point for more advanced reinforcement learning techniques. Given the potentially large action space in recommender systems (e.g., many possible items to recommend), a randomized approach can help in discovering effective strategies that might be missed by more deterministic methods.

Algorithm: Randomized Algorithm for Identifying Optimal Policies

1. best policy ← None
2. best policy performance ← $-\infty$
3. $N \leftarrow 200$
4. For i from 1 to N do:
   (a) For each $s \in S$ do:
      i. $\pi(s) \leftarrow$ random choice(UP, UD, UL, UR) // Generates a random policy
   (b) perf ← $J(\pi)$
   (c) If perf > best policy performance then:
      i. best policy performance ← perf
      ii. best policy ← π
5. return best policy

2.3.2 Value Iteration

Value Iteration, a fundamental algorithm in reinforcement learning and dynamic programming is used to compute the optimal policy for a Markov Decision Process (MDP). It focuses on finding the optimal value function, which is a function that gives the maximum value (expected return) that can be obtained from each state and It involves iteratively updating the value of each state. The value of a state is updated based on the expected return of the best action that can be taken from that state and the updates are based on the Bellman Optimality Equation, which provides a recursive way to calculate the value of a state based on the values of the neighboring states.

The key intuition behind this algorithm is that the values of states will eventually converge to the optimal values after a sufficient number of iterations and it can effectively deal with the stochastic nature of the environment (uncertainty in state transitions) and integrates the concept of delayed rewards.

In a Recommender System MDP using Reinforcement Learning, Value Iteration provides a robust framework for learning the optimal recommendation strategies in a recommender system modeled as an MDP. Its focus on learning state values and deriving policies from these values makes it a powerful tool for optimizing user experiences in dynamic and uncertain environments. However, practical implementation in large-scale systems may require adaptations or enhancements to handle complexity and scalability.

Algorithm: Value Iteration

1. Initialize v_0 arbitrarily.
2. For $i = 0$ to ∞ do:

(a) /* *Policy Evaluation* */
For all $s \in S$:

$$v_{i+1}(s) = \max_{a \in A} \sum_{s' \in S} p(s, a, s') \left(R(s, a) + \gamma v_i(s')\right)$$

(b) /* *Check for Termination* */
If $v_{i+1} = v_i$ then:

i. Terminate.

2.3.3 Temporal Difference Learning

Temporal Difference (TD) Learning is a blend of Monte Carlo ideas and dynamic programming (DP) methods. In Monte Carlo methods, learning happens from complete sequences, while in DP, learning is based on existing estimates. TD Learning can learn directly from raw experience without a model of the environment's dynamics.

The key intuition behind this algorithm is that it is a boostrap method which can update estimates based in part on other learned estimates, without waiting for a final outcome (unlike Monte Carlo methods). This is done using the TD error, a difference between estimated values at successive time steps. It can learn before knowing the final outcome, making it faster and more suitable for situations where the final outcome is not immediately known.

In a Recommender System MDP using Reinforcement Learning, TD Learning can be used to predict future user engagement or satisfaction based on current interactions, adjusting recommendations dynamically. By continuously updating the value function (which represents the expected reward or utility of recommendations), the system can adapt to changing user preferences and behaviors.Since TD Learning doesn't require waiting for the end of an episode (like a complete user session), it can update recommendations in real-time based on immediate user feedback.

Algorithm: TD Learning

---

1. Initialize $V(s)$ arbitrarily for all $s \in \mathcal{S}$, except that $V(\text{terminal}) = 0$.
2. Repeat (for each episode):
   (a) Initialize s.
   (b) Repeat (for each step of episode):
      i. Choose a from s using policy derived from V (e.g., ϵ-greedy).
      ii. Take action a, observe reward r, and next state s'.
      iii. Update V:
      $$V(s) \leftarrow V(s) + \alpha \left[r + \gamma V(s') - V(s)\right]$$
      iv. $s \leftarrow s'$.
   (c) until s is terminal.

2.3.4 Tabular SARSA

Tabular SARSA (State-Action-Reward-State-Action) is an on-policy algorithm that learns a policy based on the actions taken, as opposed to off-policy methods like Q-learning which learn from actions that might be different from the current policy.

The key intuition behind this algorithm is that it is learns the value of actions directly from the experience of following a policy. It updates the value of a state-action pair in its table based on the state it's in, the action it takes, the reward it receives, the next state it ends up in, and the next action it takes.

In a Recommender System MDP using Reinforcement Learning, Tabular SARSA provides a framework for learning from interactions in a way that is directly applicable to the dynamic and personalized nature of recommender systems, making it a valuable tool in the arsenal of reinforcement learning techniques for these applications.While tabular methods can face challenges in scaling to large state or action spaces (like those in complex recommender systems), they provide a foundational understanding and can be extended or modified with function approximation techniques for larger-scale applications.

Algorithm: Tabular SARSA

---

1. Initialize $q(s, a)$ arbitrarily.
2. For each episode do:
   (a) $s \sim d_0$.
   (b) Choose a from s using a policy derived from q (e.g., ϵ-greedy or softmax).
   (c) For each time step, until s is the terminal absorbing state, do:
      i. Take action a and observe r and s'.
      ii. Choose a' from s' using a policy derived from q.
      iii. Update q:
      $$q(s, a) \leftarrow q(s, a) + \alpha \left(r + \gamma q(s', a') - q(s, a)\right)$$
      iv. $s \leftarrow s'$.
      v. $a \leftarrow a'$.

2.3.5 Q Learning

Q-Learning is an off-policy algorithm used to find the optimal action-selection policy for any given finite Markov Decision Process (MDP).It learns the value of the optimal policy independently of the agent's actions and it can also learn from actions that are outside the current policy, such as random exploratory actions. Here, Q-values are updated using the Bellman equation.

The key intuition behind this algorithm is that it aims to learn the value of the best possible action in each state in a complex dynamic environment, without needing a model of the environment.Also, it can effectively handles situations where rewards are delayed, which is a common scenario in many real-world problems.

In a Recommender System MDP using Reinforcement Learning, Q-Learning learns recommendations that are most effective in different states (user contexts) which can help in

personalizing the recommendations to individual users. Q-Learning is well-suited for environments where user behavior is stochastic and can change over time, as it continually updates its policy based on ongoing interactions. It can help in effectively navigating the trade-off between exploring new recommendations and exploiting known preferences to enhance user experience and engagement.

Algorithm: Q-Learning

1. Initialize w arbitrarily.
2. For each episode do:
   (a) $s \sim d_0$.
   (b) For each time step, until s is the terminal absorbing state, do:
      i. Choose a from s using a policy derived from q.
      ii. Take action a and observe r and s'.
      iii. Update w:

$$w \leftarrow w + \alpha \left(r + \gamma \max_{a'} q_w(s', a') - q_w(s, a) \right) \frac{\partial q_w(s, a)}{\partial w}$$

      iv. $s \leftarrow s'$.

3 Analysis and Results

3.1 Plots from Fine Tuning of Hyper-Parameters

To perform Hyper-parameter tuning we have exhausitively used Grid Search over all the hyper-parameters which are and β - weights applied to all transitions if the recommender's recommendation was accepted or rejected respectively. Additionally, we've hyper-parameters k and γ that denotes the k last purchases made by the user and discount rate respectively.

To perform grid search, we've taken permutations of boundary values close from [0, 1] for params , β and γ. Similarly, we've varied k to be in range [1,6] to understand how the user purchase history influences the decisions of the recommendation system.

Please refer to figure 2 and Discussions section for plots and analysis on Hyper-parameter tuning.

3.2 Evaluation

3.2.1 Plots from Metric: Recommendation Score

To assess accuracy, a recommendation is considered successful when the observed item t_i ranks within the top m recommendations (where m ranges from 1 to 10). The success rate, denoted as RC, represents the percentage of cases where the prediction is accurate. A score of 100 indicates success in all instances. Optimal hyper-parameters were employed for all reinforcement learning (RL) algorithms to generate recommendation scores for the top 1 to 10 recommendations. Refer to figure 3 for graphical representations and the Discussions section for detailed analysis.

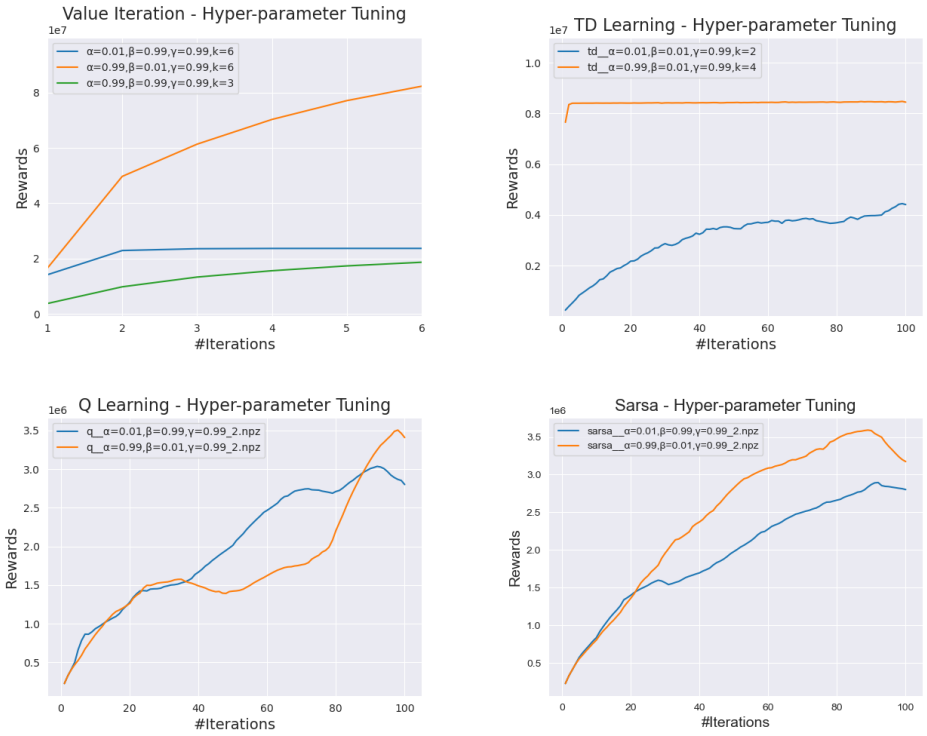


Figure 2: Key Results from Hyper-Parameter Tuning

3.2.2 Plots from Metric: Exponential Decay Score

The accuracy metric relies on the placement of the observed item within the recommendation list, operating under the assumption that recommendations appearing higher on the list are more likely to be viewed by users. Please consult the plot labeled as Figure 4.

3.2.3 Plots for Metric: MDP Reward Function

As mentioned in the Reward subsection of the Markov Decision Process (MDP), the MDP's reward is defined as the ratio of the aggregate time spent playing all games in state s to the cumulative sum of prices of all games. Optimal hyper-parameters have been employed to generate reward versus Iteration graphs for k values ranging from 1 to 4, providing insights into how the reward is influenced by the last k purchases. Please consult figures 5, 6, 7, and 8 for visual representations of these results.

3.3 Discussions

- **Hyper-parameter Tuning -** From Figure 2, we can see few common patterns that can be observed across all analyzed reinforcement learning (RL) algorithms.

  – **User's Last K Purchases (denoted as k) -** As depicted in Figure 2, the augmentation of the parameter K correlates with an increase in the reward. This aligns with intuition,

considering that the Markov Decision Process (MDP) lacks negative rewards, and each state is indicative of the cumulative rewards of individual items. This observation is reinforced by the graphical representation of iteration versus reward.

- **Weight given to items purchased by the recommendation of recommender system (denote as α) -** Analysis of hyper-parameter tuning plots (refer to Figure 2) indicates that elevated values of $\alpha = 0.99$ result in more gradual curves and a delayed stabilization of rewards. To prioritize the agent's "exploitation" of the Markov Decision Process (MDP), we have selected the optimal value of $\alpha = 0.75$ for conducting our benchmark experiments.

- **Weight given to items purchased against the recommendation of recommender system (denote as β) -** The hyper-parameter tuning plot (Figure 2) reveals that elevated β values result in sub-optimal policy convergence, as they prioritize actions conflicting with the recommender system. Our objective is to encourage the agent to explore and provide recommendations that, on the whole, yield better outcomes than the next best greedy recommendation. Therefore, in our benchmark analysis, we opted for $\beta = 0.25$ to strike a balance.

- **Discount Rate (represented as γ) -** While we explored a range of discount rates from 0.01 to 0.99, we observed minimal impact on the convergence of policies to optimal states. Setting γ to 0.01 resulted in slightly faster convergence; however, in the majority of cases, we converged to the same optimal policy. Therefore, we opted for γ=0.99 for conducting our experiments.

- **RL Algorithms Convergence** - Here we discuss the convergence of all RL algorithms -

  - **Randomized Algorithms for Optimal Policies** The results depicted in Figure 5 demonstrate convergence of this approach to the optimal policy for states k=2 to 4. Notably, the curve exhibits a smoother trajectory at k=2, whereas it quickly plateaus (10-20 iterations) for k > 2.

  - **Value Iteration**: Illustrated in Figure 6, the algorithm converges within 15 iterations for k = 1 or 2. For k > 2, it shows faster convergence with elbow curves and a corresponding increase in optimal reward as k grows.

  - **Temporal Difference (TD) Learning**: From Figure 7, a distinctive hill-shaped curve is observed for k=2, initially converging to a sub-optimal policy and subsequently reaching the optimal policy after 80 iterations. In contrast, for k>2, the algorithm achieves convergence to the optimal reward within a mere 5-10 iterations.

  - **Q Learning and SARSA Convergence Patterns** : Analysis of the convergence patterns depicted in figures 8 and 9 reveals similar trends. Specifically, when k=1, the algorithms exhibit a propensity for greedy steps towards the optimal policy. As k increases to 2, both algorithms display smooth elbow curves, indicative of a gradual convergence. Beyond k=2, it becomes evident that the policies converge to optimal rewards within 5-10 iterations.

- **Recommendation Score** - As observed from figure 3 , Value iteration (k=4) and TD learning (k=1) algorithms perform much better than the other algorithms . It can be observed

that more the k better is the performance of value iteration because its focus on learning state values and deriving policies from these values makes it a powerful and robust tool for optimizing user experiences in dynamic and uncertain environments like our Recommender System MDP. In case of TD Learning it can update recommendations in real-time based on immediate user feedback, it doesn't require waiting for the end of an episode (like a complete user session) hence it works best even at k =1 to get the best-known actions to maximize rewards.

- **Exponential Decay** - As observed from figure 4 , for Value iteration (k=3 and 4), Randomized Algorithms (k=3 and 4),Q Learning (k=3 and 4) and Tabular SARSA (k=3 and 4) - more the value of k better is the model fore predicting recommendations. This happens for the simple reason that most algorithms require space to find a trade off for exploring and exploiting the best-known actions to maximize rewards. (Q Learning has some outliers) However in case of TD Learning it can update recommendations in real-time based on immediate user feedback, it doesn't require waiting for the end of an episode (like a complete user session) hence it works best even at k =1 to get the best-known actions to maximize rewards.

4 Conclusion

4.1 Final Thoughts

In summary, this research effectively applies the Markov Decision Process (MDP) based on the methods outlined in the cited reference paper [1]. Various Reinforcement Learning techniques, including Randomized Algorithms for Optimal Policy, Value Iteration, TD Learning, Q Learning, and SARSA, are employed on a synthetic dataset obtained from Kaggle. The study includes hyper-parameter tuning for parameters such as $alpha$, *beta*, *gamma*, and k. Evaluation metrics proposed in the reference paper, specifically the Recommendation Score and Decay Score, are used to assess outcomes. A comparative analysis of a custom reward function against the provided dataset highlights the individual performance of each algorithm.

The Recommendation Score Plot [Figure 3] reveals that Value Iteration performs exceptionally well when evaluated against the last 4 user purchases. Additionally, both TD Learning and Value Iteration converge to the optimal policy with maximum reward. Notably, TD Learning demonstrates robustness, converging to optimal policies even with low values of k, in contrast to Value Iteration. This resilience arises from its ability to update the policy dynamically in real-time, without waiting for the episode to conclude. TD Learning proves particularly well-suited for infinite horizon Markov Decision Processes.

4.2 Future Scope

In future, we'd like to work on enhancing the resilience of our Markov Decision Process (MDP) through the incorporation of a more diverse dataset, thereby achieving a balance between exploration and exploitation. Further, we'd like to test our MDP against more advanced Deep Reinforcement Learning (RL) algorithms such as Action Critic, PPO, and many others.

In order to bolster scalability, our focus will extend to introducing stochastic elements into the MDP. This will involve diversifying the factors considered in user recommendations, encompassing variables like budget, user preferences, demographics, and more.

Finally, we'd like to work on real-world deployment, involving real-time interactions with users. This empirical approach will provide valuable insights into the performance of our system under authentic conditions, allowing us to validate and optimize its practical utility.

4.3 Source Code and Contribution

Both Authors have contributed equally towards this research. Source Code is available at Github[2] and the dataset is curated from Kaggle[3]

References

[1] G. Shani, D. Heckerman, R.I. Brafman, C. Boutilier, Journal of Machine Learning Research **6** (2005)

[2] Y. Li, K. Liu, R. Satapathy, S. Wang, E. Cambria, arXiv preprint arXiv:2306.12680 (2023)

[3] M. Fouladirad, J. Neal, J.V. Ituarte, J. Alexander, A. Ghareeb, Int J Data Anal Inf Syst **10**, 13 (2018)

[4] B. Marr, *Artificial intelligence in practice: how 50 successful companies used AI and machine learning to solve problems* (John Wiley & Sons, 2019)

[5] M. Yesilada, S. Lewandowsky, Internet policy review **11** (2022)

[6] M. Millecamp, N.N. Htun, Y. Jin, K. Verbert, *Controlling spotify recommendations: effects of personal characteristics on music recommender user interfaces*, in *Proceedings of the 26th Conference on user modeling, adaptation and personalization* (2018), pp. 101–109

[7] S.C. Geyik, S. Ambler, K. Kenthapadi, *Fairness-aware ranking in search & recommendation systems with application to linkedin talent search*, in *Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining* (2019), pp. 2221–2231

[8] A.S. Das, M. Datar, A. Garg, S. Rajaram, *Google news personalization: scalable online collaborative filtering*, in *Proceedings of the 16th international conference on World Wide Web* (2007), pp. 271–280

[9] M. Howe, A Case Study, I pp. 1–6 (2009)

[10] Y. Hu, Y. Koren, C. Volinsky, *Collaborative Filtering for Implicit Feedback Datasets*, in *2008 Eighth IEEE International Conference on Data Mining* (2008), pp. 263–272

[11] M.J. Pazzani, D. Billsus, *Content-Based Recommendation Systems* (Springer Berlin Heidelberg, Berlin, Heidelberg, 2007), pp. 325–341, ISBN 978-3-540-72079-9, `https://doi.org/10.1007/978-3-540-72079-9_10`

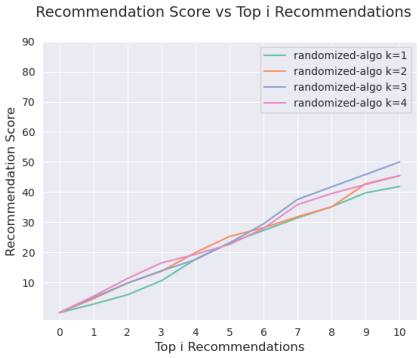
[12] Y. Koren, R. Bell, C. Volinsky, Computer **42**, 30 (2009)

[13] P. Melville, R.J. Mooney, R. Nagarajan, *Content-Boosted Collaborative Filtering for Improved Recommendations*, in *Eighteenth National Conference on Artificial Intelligence* (American Association for Artificial Intelligence, USA, 2002), p. 187–192, ISBN 0262511290

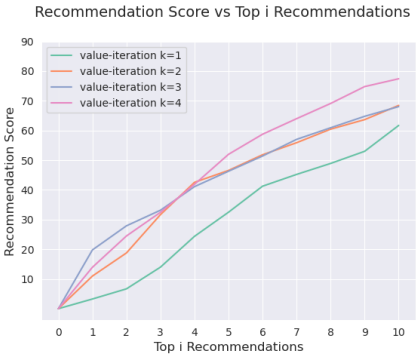
[14] D.N. Chen, P.J.H. Hu, Y.R. Kuo, T.P. Liang, Expert Systems with Applications **37**, 8201 (2010)

[2] https://github.com/singh96aman/Recommender-Systems-with-Reinforcement-Learning

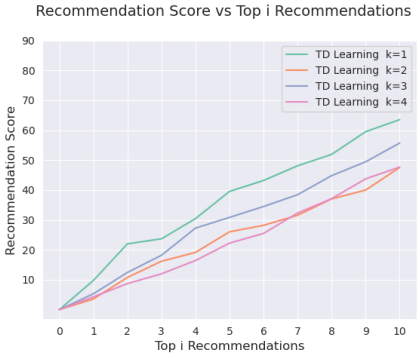
[3] https://www.kaggle.com/datasets/arnabchaki/popular-video-games-1980-2023



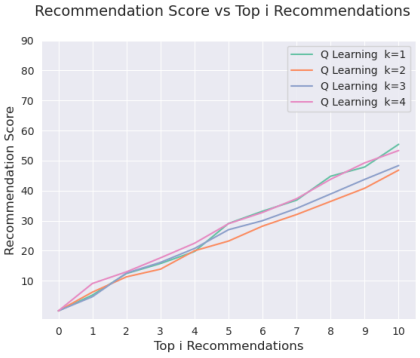
(a) Randomized Algorithm



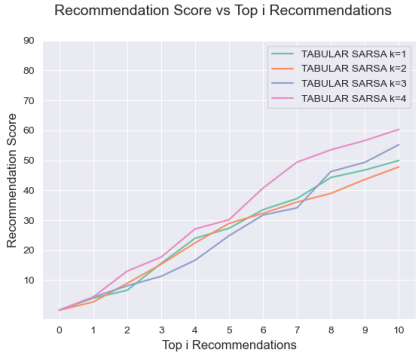
(b) Value Iteration



(c) TD Learning

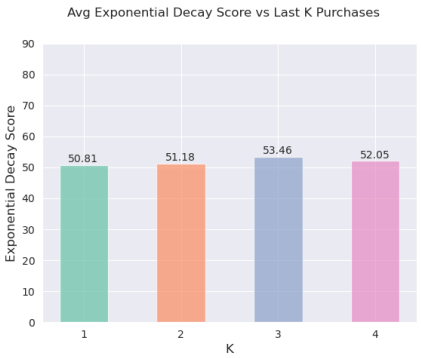


(d) Q Learning

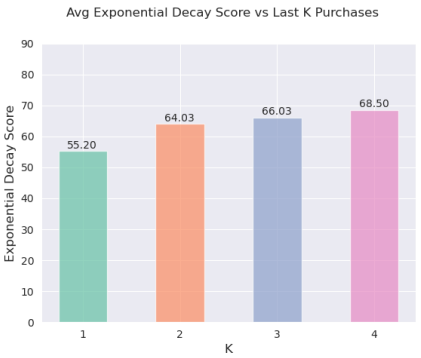


(e) Tabular SARSA

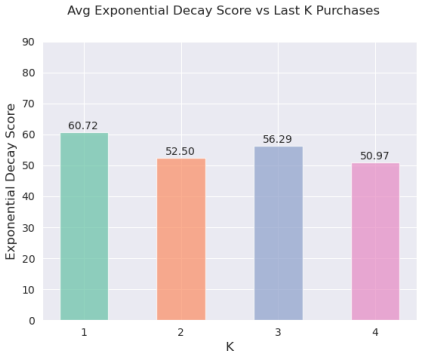
Figure 3: Recommendation Scores of RL Algorithms on Recommender MDP



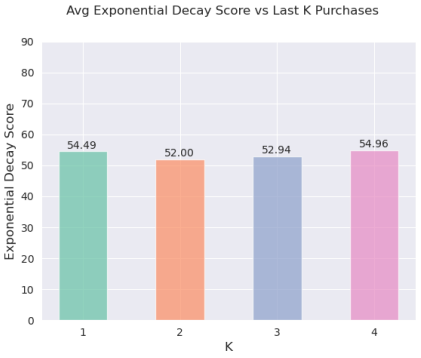
(a) Randomized Algorithm



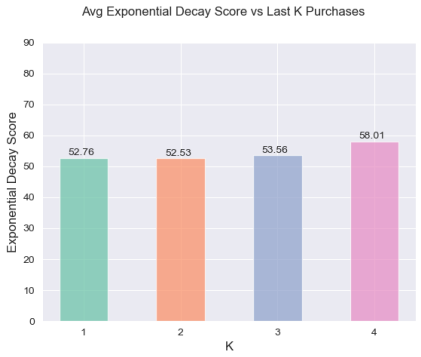
(b) Value Iteration



(c) TD Learning



(d) Q Learning



(e) Tabular SARSA

Figure 4: Exponential Decay Scores of RL Algorithms on Recommender MDP

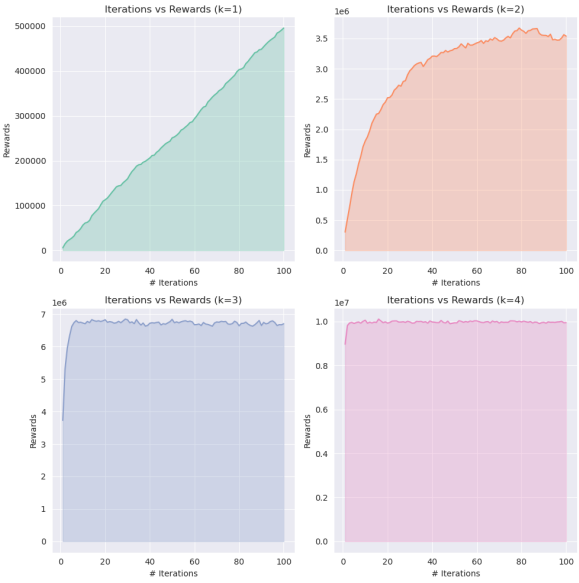


Figure 5: Randomized Algorithm

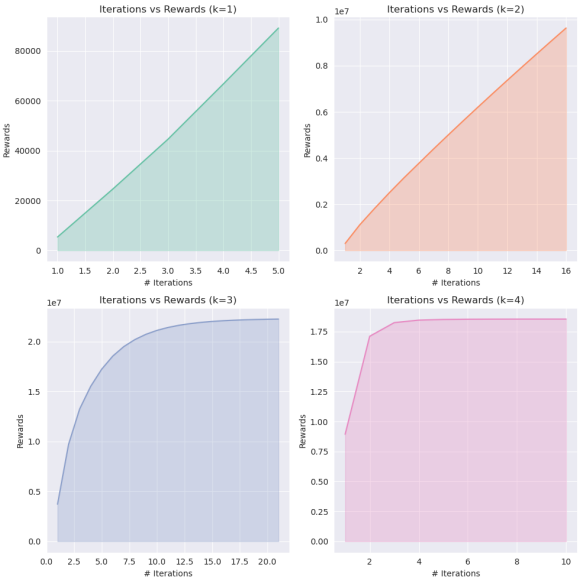


Figure 6: Value Iteration

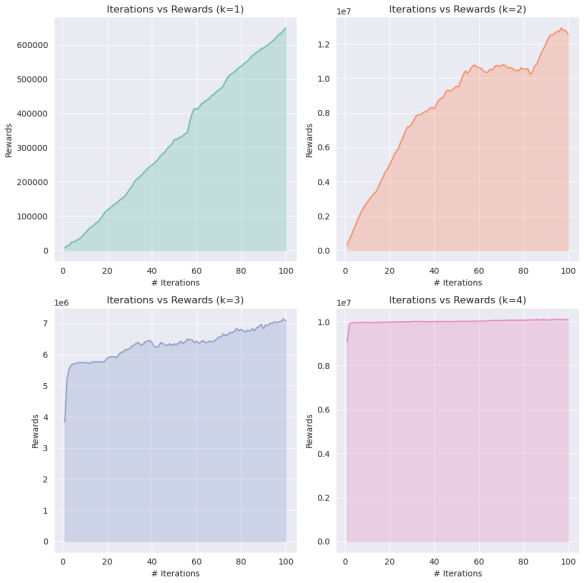


Figure 7: TD Learning

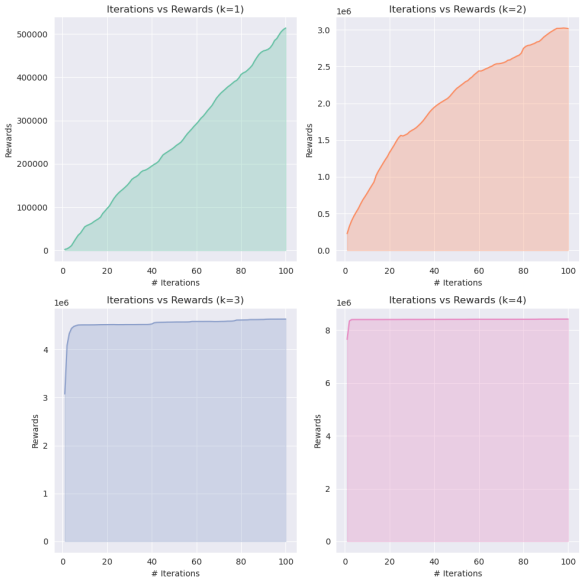


Figure 8: Q Learning

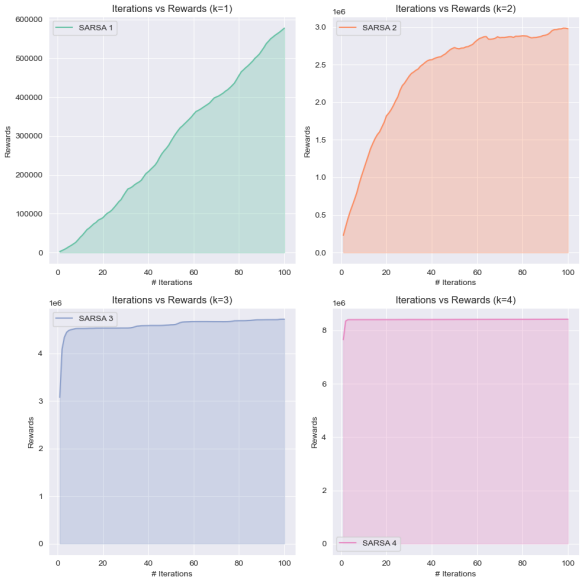


Figure 9: SARSA