Slate Recommendation Methods

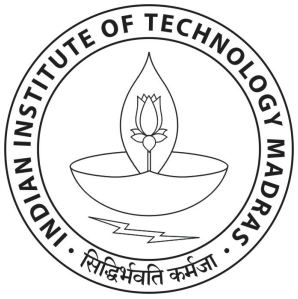
A Project Report

submitted by

KSHITIJ SHAH

in partial fulfilment of requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

MAY, 2023

THESIS CERTIFICATE

This is to certify that the thesis titled **Slate Recommendation Methods**, submitted by **Kshitij Shah**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. B. Ravindran
Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Dr. Nandan Sudarsanam
Associate Professor
Dept. of Management Studies
IIT-Madras, 600 036

Place: Chennai

Date: April 21, 2023

ACKNOWLEDGEMENTS

I would like to thank Prof. Nandan Sudarsanam and Prof.B. Ravindran for their guidance and assistance, from helping me frame the problem statement to obtaining working models and overcoming the many obstacles encountered along the way. I would also like to take the time to thank them for helping me with the issues I faced with limited research in this area.

I would also like to thank Mr. Rajesh Barnwal, Mr. S Karthick Raja, Mr. Vinod Kumar Tirupati, and Mr. Manikandan Radhakrishnan for their assistance, regular input, and expertise in this area. Completing this project would have been very difficult without their input and added effort.

Finally, I would like to thank my Faculty Advisors, Prof. Hema Murthy, and Prof. Madhu Mutyam for their assistance in the completion of this project as well as my B.Tech Degree.

ABSTRACT

KEYWORDS: Slate Recommendation Systems, Combinatorial Action Spaces

Recommender Systems have become a crucial leg in today's world of personalized digital assistants. While significant research has been carried out to understand user preferences, and search engine optimization, there is not enough research into providing users with a slate of products. In effect, the slate changes the possible action space to a combinatorial set. Standard Reinforcement Learning Techniques fail to develop good results on this combinatorial action space. To change this, an alternative way to evaluate the quality of slates during learning and training has to be created and an alternate method of evaluation to cycle through a large number of possible combinations while presenting to the user. We apply state-of-the-art methods to this process to break down the learning of a combinatorial sets into a linear number of items instead. We also experiment with non linearizing the process using Neural Networks. For the Evaluation mechanism, we look into using four different methods that are simpler including a couple of greedy mechanisms, a neural network and mixed integer programming. Besides this, we look into using Linear UCB Bandit Algorithms and Neural UCB Bandit Algorithms in this domain since they are now being used in Recommender System Environments and could provide good results here as well.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology, Madras
RL	Reinforcement Learning
IEE	Interest Evolution Environment
IXE	Interest Exploration Environment
LTS	Long Term Satisfaction Environment
LTVs	Long Term Values
MIP	Mixed Integer Programming
CTR	Click Through Rate
SoTA	State-of-the-Art
CMAB	Combinatorial Multi Arm Bandits

NOTATION

n	Number of items in the document corpus
k	Number of items in slate
A	Action chosen i.e slate shown to the user
i	Item chosen i.e item user interacts with
$\mathcal{A}$	Action set: Including all items in the corpus
$v(x_{ij})$	Probability function
(Outputs the probability of a user i choosing item j based on a user item feature vector x_{ij})	
s	Current state of the user
$s1$	Next state of the user
$R(s, A)$	Reward obtained when choosing action A in state s

CHAPTER 1

INTRODUCTION

1.1 Introduction to RL in Recommender Systems

Reinforcement Learning (RL) has recently grown as an effective method of improving Recommender systems. Instead of the usual methods such as collaborative filtering (Wenga *et al.* (2021)) or content-based filtering (Glauber and Loula (2019)) that have limitations in terms of scalability , personalization, and diversity of recommendations, RL systems offer an alternative. RL agents learn through reward systems, which are based on user feedback and satisfaction. This makes them much more likely to be able to scale the problems faced by other methods.

RL-based Recommender systems usually employ a Markov Decision Process to model the recommendation setup and help it learn a policy to map the user's context and history to recommendations that the model might give. It then updates the policy to maximize expected reward.

One of the challenges that an RL agent faces is balancing the explore-exploit trade off. In effect, this means that the recommender systems is balancing between recommending popular items (exploitation) and recommending less popular items that could be a better fit for the user's preferences (exploration). Exploration strategies such as epsilon-greedy, Thompson sampling, and Upper Confidence Bound (UCB) algorithms have been proposed to address this challenge.

RL-based recommender systems have shown promising results in various domains, such as online advertising, music, and movie recommendations, and personalized news recommendations. However, several research questions such as temporal dynamics, domain knowledge incorporation, and handling multiple objectives - where we conduct our research, remain unanswered.

1.2 Introduction to Slate Recommender Systems

Slate recommendation systems have emerged as a new type of recommender system that recommends a collection or slate of items, rather than a single item, to users. Unlike traditional recommender systems that typically recommend items based on user preferences or past interactions, slate recommendation systems consider the interaction between items in the slate and the user's preferences to provide a set of items that collectively provide the highest user satisfaction - this leads to combinatorial action spcaes and it is imperative to develop new methods to learn in this setup since standard models fail with such large action sets.

Slate recommendation systems are formulated as multi-objective optimization problems that aim to optimize various metrics such as diversity, novelty, and coverage while maximizing user satisfaction. Most research uses methods like matrix factorization that can be employed to factorize the user-item interaction matrix into two matrices, one for users and one for items. The resulting matrices are then used to generate a slate of items that are likely to be preferred by the user.

However given that RL has shown success with other Recommender Systems, the natural question arises: Can we adapt it to work here? Slate recommendation systems have demonstrated success in several domains, including e-commerce, music, and video recommendations. However, challenges remain in handling user preferences over different dimensions, scalability issues, and balancing the trade-off between user satisfaction and other objectives.

1.3 Project Objectives

The objectives of this B.Tech Project are as follows:

1. Set up a suitable environment to study slate recommendation systems.
2. Explore combinatorial action spaces and test traditional Reinforcement Learning Methods on them.
3. Study the current state-of-the-art methods in Slate Recommendation and test them against traditional Reinforcement Learning Algorithms.
4. Explore the possibility of using alternative methods that could factor in combination data into item and slate values.

5. Look into the effect that combination values have on the outcome and slates selected by algorithms.

6. Conclude as to how progress could be made to factor in combination data, and what agents are the best fit for the slate recommender problem.

CHAPTER 2

LITERATURE SURVEY

RL for Recommender Systems is mainly used to replace earlier methods like Content-Based Filtering (Glauber and Loula (2019)) and Collaborative Filtering (Wenga *et al.* (2021)) which helps solve the issues of having cold starts as well as better exploitation functions in cases of changing action sets. to understand related work. A recent work that studies the surveys the scope of RL in Recommender Systems (Lin *et al.* (2021)) talks about how current RL systems suffer from issues such as Data Scarcity and Biased Recommendation, which is a problem we look to address here.

There is some work into Combinatorial Multi-armed Bandits (Chen *et al.* (2014)) which involves the creation of 'super-arms' or 'super-actions' which are composite actions. The authors work on presenting multiple items to the user. However, a key difference is that the authors look to do this sequentially - one at a time - unlike a slate where all items are presented together. However, this motivates a look into using bandit algorithms for the Slate Recommendation Problem as well. Here another issue is that the rewards obtained are also obtained against the whole slate of actions presented rather than items off the slate

Another approach is using User Choice Models into which there is once again, extensive research that looks to understand simple MNL models and optimize them to model user behaviour. If we are able to leverage a strong User Choice Model, then it could be used to solve the Slate Problem as proposed in the Slate Q Decomposition (Ie *et al.* (2019*b*)). This uses MNL models (Li *et al.* (2020)) and Click-through Rate Optimizers. (Joachims (2002)) These MNL models study object features and user interactions - however, a significant drawback is that they do not study objects together. This limits their use when trying to understand item interactions on slates.

Another avenue of research that is relevant to our direction of study is ranked systems - for example, cascading bandits (Kveton *et al.* (2015)) has been used to sequentially present a list of ranked documents to users and significantly outperforms learning-to-rank algorithms. They provide 2 algorithms - CascadeUCB1 and CascadeKL-UCB which both have an upper bound on regret. However, a limitation of this method is

its scalability. It is not equipped to handle a large document corpus without feature embedding.

There is also research into how the placement of items on a ranked list affects their likelihood of being chosen. (Gao *et al.* (2018)) This suggests that we could study if there is a significant spot on the slate more likely to be clicked - for example, the centre spot and assign a weight to it. While the paper focuses on the placement in a list form, it does propose an algorithm we could bring in to use to learn separate weights for every spot on the slate in case there seems to be a difference. This is likely to be limited to just the central spot having a higher likelihood of being chosen as compared to the remaining spots in a grid-like setup.

Now, specifically in the domain of Slate Recommender Systems, we first look at work in Off-Policy Evaluation that harness CMABs but look to reduce the amount of data required to train such an algorithm. While previous work looks into

1. Restricting attention to small slate spaces
2. Partial matches between the proposed and observed slates
3. Assuming that the policies used for logging and evaluation are similar

This is all done to combat the issue that the policy being evaluated is likely to choose different slates from those recorded in the data. These algorithms in the past have shown high bias and so to counter that issue, the authors have proposed a new pseudo inverse estimator that assumes that that the slate-level reward is a sum of unobserved action-level rewards and use that to guide the algorithm.

Alongside this, we also look into user choice models that have been proposed to theorize the probability of choosing a certain item when a slate is proposed to a user. This model could give us access to probabilities that could be used to break down the slate problem. We look into MNL models and related work in this field. (Li *et al.* (2020))

We then look into Non-Linear approaches to the Slate Recommendation Problem. These include works in Neural Contextual Bandits and NeuralUCB (Kassraie and Krause (2022), Zhou *et al.* (2019)). Here, the algorithm has two components: a linear contextual bandit model that models the relationship between the context and the reward, and a neural network that captures the complex relationships between the context and the actions. We look to explore this further here.

Srinivas *et al.* (2009) Swaminathan *et al.* (2016) Covington *et al.* (2016)

CHAPTER 3

CHOSEN ENVIRONMENT:
Google Recsim's Interest Evolution

3.1 Introduction

We need a large dataset to handle the combinatorial action space along with a reward space that can work out the value of slates beyond simple selection while learning. Google RecSim (Ie *et al.* (2019*a*), Mladenov *et al.* (2021)) is a research framework for developing recommendation algorithms using simulation instead of static databases. It allows a user to simulate interactions with a recommendation system, enabling them to evaluate the performance of different algorithms in a controlled environment. This allows for researchers to test, improve, and deploy algorithms without doing one of the two: either partaking in real world experiments with every change made to the algorithm or second, working with a static data set, that does not allow the algorithm to learn as well as it could. In the latter case, it could be that a certain action i.e a certain slate would be ideal based on the exploration-exploitation algorithm behind the agent, to show as it's next training step but unless that very slate has been shown earlier, it would not be able to learn from that. And, especially, given that we are dealing with combinatorial action spaces, it is very likely that a certain slate has not been shown. This ensures that a simulation software is always going to be a better method of dealing with this problem than a static dataset.

Recsim is built with customisable user behaviour, reward systems, and multiple optimizable environments to test different aspects of a recommendation system.

We choose the Interest Evolution Environment out of these spaces. The reasoning behind this choice is as follows:

1. The Interest Evolution Environment (IEE) is one of three setups in RecSim including Interest Exploration (IXE), and Long-term Satisfaction (LTS).
2. IEE focuses on changes in user preferences and evolution in tastes as time passes. It models user interests and features and adapts them as the user interacts further

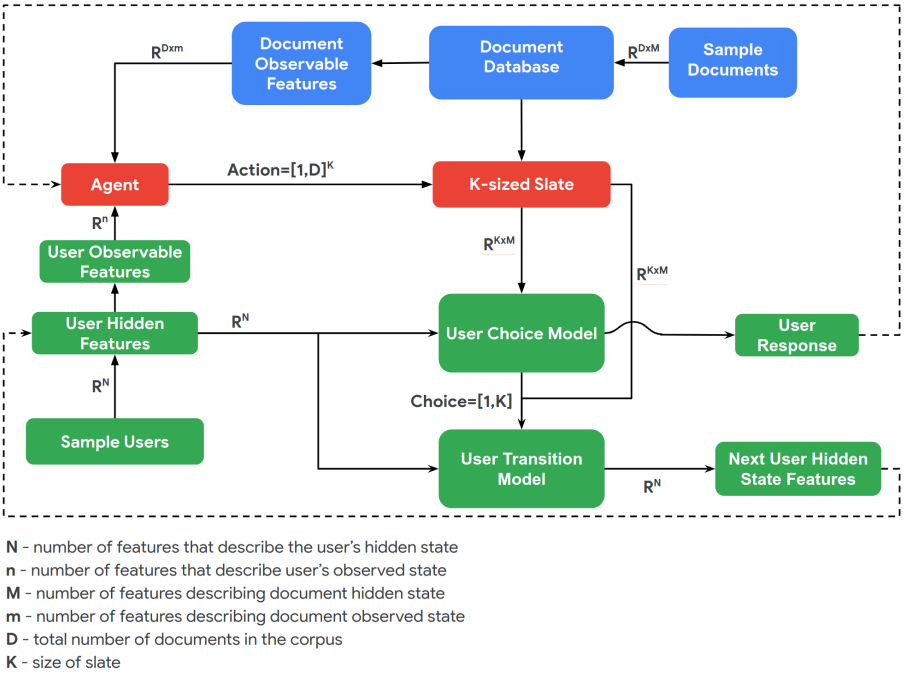


Figure 3.1: The RecSim Framework Ie *et al.* (2019*a*)

with the environment. A good example of the same would be a movie recommender or new recommendation site.

3. IXE focuses on the exploration of a state space and how user preferences can be modelled in the presence of non-familiar items. This could model a shopping website when it chooses what to display to a first time consumer.

4. LTS focuses on a specific scenario where short term gains (eg: taste) are opposed to long term benefits (eg: health). It is useful in testing whether a recommender system is myopic in it's outlook or is able to optimise for long term gains.

Here, IEE is the most common setting for a recommender system and thus was chosen as the environment we use to test our models on. (Ie *et al.* (2019*a*))

3.2 State Space

1. User - In IEE, the state space is modeled as a vector that includes features and preferences of the user, taking into account various factors such as the user's past interactions, the relevance of items, and the user's current state. The IEE uses a Markov Decision Process (MDP) to model the state space and simulate the user's behavior over time.
   The users all start with a fixed budget of time which decreases as we progress but is also piqued by good recommendations. A limited number of these features are observable and those are made available to the agent.

2. Document Observable Features - The content is characterized by a vector of features and quality of the item itself.
3. Response - This indicates the user's response to the last slate starting at 'None'. It consists of click - indicating whether video was chosen, time_watched - time spent on video, liked - whether the user liked the video and quality.

For the RL algorithm, the state space is modelled as a combination of known user features, the user's past interactions, and the user's current state.

3.3 Action Space

The action space is a slate with k items chosen out of a database of n items. The agent's behavior involves choosing and prioritizing a set of k items (or documents, as depicted in the diagram) to create a slate of recommendations for the user.

3.4 Reward Space

The reward is a single scalar value that is directly proportional to how satisfied the user is with the item they chose. This means that an item on which the user spends more time, likes, or aligns more with their interests will get higher points than an item that is scrolled past quickly, not liked, or in opposition to user interests.
The reward value in the IEE is typically updated over time as the user interacts with more recommended items. The system's goal is to maximize the cumulative reward over a sequence of user interactions, which can be modelled using a reinforcement learning framework.

CHAPTER 4

MODELS

4.1 Full Q Learning - Baseline

1. Philosophy: To start with, we set up a simple Q learning model to learn Q values for each action i.e we learn $\binom{n}{k}$ Q-values. This is a standard RL method and will act as the baseline model to compare our future models against.

2. Theory: Q-Learning is a type of reinforcement learning algorithm that aims to find an optimal policy for an agent in a Markov Decision Process (MDP). In reinforcement learning, an agent interacts with an environment to learn by trial and error to maximize a cumulative reward signal. In Q-Learning, the agent learns a Q-value function, which estimates the expected cumulative reward for taking a particular action in a particular state. The Q-value function can be represented as a lookup table or a neural network. The algorithm updates the Q-value function based on the observed rewards and the transition to the next state, using a temporal difference (TD) learning approach.

3. Formula:

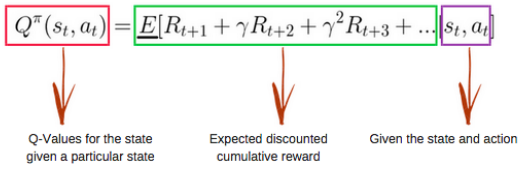


Figure 4.1: Q-Learning Formula ZADL (2019)

4. Results: We settle on a reward of 149 after 800 episodes. Interpretation of this number is as follows:

   (a) We have set a time budget of 200 to start every episode with one user.

   (b) As they watch videos, we increase the total watched time and their budget is also modified.

   (c) Average video length is 4, and the maximum possible reward in an average episode is around 250

   (d) In this case, we obtain a rolling average of 149 with Full SlateQ learning

4.2 SlateQ - Successful (SOTA)

1. **Philosophy**: The idea behind SlateQ Learning is to decompose the slate into item-wise values, use a user choice model to estimate probabilities of selection and then put together a slate based on multiple possible optimizing methods (offline and online). This makes the computation far simpler since we remove the combinatorial aspect of the action space, but on the flip side, it removes any combinatorial information except that which is stored in the user choice model itself.

2. **User choice Model**: To begin with, the SlateQ setup uses a user choice model. It is explained as follows:
   (a) Input: Interaction Data from users
   (b) Output: The user choice model outputs a relative probability of selecting each item for every grid. For example, this probability would be 1/10 for each item before training. (The 10th item being a null item). After that, it would then train to calculate relative probability.
   (c) How: SlateQ does not explain this model. It lists multiple possible models we could use here. For the purpose of our model, we use a Multinomial Logit choice model. This model is simple in that it calculates a value for each item being picked as the number of times the item was chosen out of the times the item was shown in the slate. This results in a value between [0,1]. It then normalizes these values for the nine items chosen in the slate to give a final probability for each item.
   (d) Note: An alternate model was also chosen here where we added a probability for the not chosen action as well. However, this only reduced the value of each slate significantly since many slates had no interactions in the data-set. Thus, this value was kept aside.

Figure 4.2: User Choice Model Ie *et al.* (2019*b*)

$$P(j|A) = v(x_{ij}) / \sum_{l \in \mathcal{A}} v(x_{il})$$

Here the user i selects item $j \in A$ with unnormalized probability $v(x_{ij})$, where v is some function of a user-item feature vector x_{ij}

3. **Decomposing Slate Q Values**:
   (a) Here the state S reflects the user state, features, history or user's past behavior.

(b) The actions A are possible recommendation slates. The items are listed as I, so any possible action is a subset of I s.t. |A| = k, where k is the slate size.

(c) P (s1 | s, A) is the transition probability from state s to state s1 after taking action A.

(d) Reward R(s, A) is the expected reward of a slate.

(e) The Bellman Equations are as follows:

$$V^*(s) = \max_{A \in \mathcal{A}} R(s, A) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, A) V^*(s')$$

$$Q^*(s, A) = R(s, A) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, A) V^*(s')$$

These are the standard bellman equations we look to maximize

(f) We can use multiple models to optimize the Bellman Equation:

i. **SARSA Algorithm**

ii. $\epsilon-$**Greedy Q Learning**

(g) The decomposition that SlateQ now executes to decompose LTVs for the slates into the items themselves is as follows. We start with two assumptions:

i. Single Choice - The user can choose only one item from the slate.

ii. Reward/transition dependence on selection (RTDS): The realized reward (user engagement) and the state transition depend only on the item consumed by the user.

(h) The assumptions allows for the following breakdown of reward values of the slate into each item

$$R(s, A) = \sum_{i \in A} P(i|s, A) R(s, i)$$

$$P(s'|s, A) = \sum_{i \in A} P(i|s, A) P(s'|s, i)$$

(i) Now this leads to Q-values being calculated as follows:

$$\overline{Q}^{\pi}(s, i) = R(s, i) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, i) V^{\pi}(s')$$

$$Q^{\pi}(s, A) = \sum_{i \in A} P(i|s, A) \overline{Q}^{\pi}(s, i)$$

This is the calculation for each item's Q-value, and then putting it together for the slate.

(j) This basically means that we calculate individual item Q values instead of calculating action values for the slates, which can be done using simple TD learning since the action space drops significantly in size.

4. **Slate Selection Models**: Now that we have calculated the Q values for the item, the next step would be to put them together to choose the best slate for display.

There are 4 possible models that are used. The first three models are used are suggested by the SlateQ paper itself, and the last one is an additional attempt at finding a better model. (Ie *et al.* (2019*b*))

(a) **Top-k**
This slate selection method uses the k items that have the highest Q-values directly to create a slate for the user. This method is the fastest in execution and leads to the fastest computation. It can be used for online slate selection to present to users so as to give quick computations.

(b) **Greedy Step**
This slate selection method adds one item to the slate with the highest Q-value, followed by then choosing the item that maximizes the slate value instead of just the highest Q-value item. The difference in the slate value vs the highest Q-value comes from the fact that the probabilities also change from the user choice model depending on what item is added to the mixture. This leads to an overall optimized slate - and may lead to creation of a slate with one very attractive item, followed by numerous unattractive items in order to boost the chances of selection of the first item itself. This is a slight downgrade on the computation time from the Top K mechanism:

$$\arg\max_{i \notin A} \frac{v(s,i)\overline{Q}(s,i) + \sum_{l<L} v(s,i_{(l)})\overline{Q}(s,i_{(l)})}{v(s,i) + v(s,\perp) + \sum_{l<L} v(s,i_{(l)})}$$

This is the calculation for the next item in the greedy calculation.

(c) **Mixed Integer Programming**
Here we use a mathematically accurate way of determining the best 9 items on the slate. It uses the following ILP optimization, which can be solved in polynomial time. This program ensures a maximal solution in polynomial time that can be implemented offline. This method compromises on the computation time to give the best results.

$$\max_{\substack{A \subset \mathcal{I} \\ |A|=k}} \sum_{i \in A} P(i|s,A)\overline{Q}(s,i)$$

$$max \sum_{i \in \mathcal{I}} \frac{x_i v(s,i)\overline{Q}(s,i)}{v(s,\perp) + \sum_j x_j v(s,j)}$$

$$s.t. \sum_{i \in \mathcal{I}} x_i = k; x_i \in [0,1], \forall i \in \mathcal{I}$$

This is the framing for the Integer Linear Program

(d) **Neural Network**
Finally, we can use a Neural Network to help choose the optimal k actions based on the input, including the user features and Q-values of the items and then choose nine items in the slate. This could work as an offline and online method and ideally would present results at the same level as Mixed Integer Programming.

5. **Placement of Actions on Slate**
Along with this, there is one further consideration - regarding the placement of actions on the slate i.e. ordering of the chosen actions. There is some research

into ordered search results that we use to decide how to place the actions in the slate. Studying the selection patterns in general, we adopt a rule-based placement which suggests the best actions be placed linearly from 1st to last (Swaminathan *et al.* (2016))

6. **Experimental Setup**
   We choose the number of items $(n) = 10$, and the number of items on the slate as $(k) = 2$ to process the computation with limited computational power. We choose the $\epsilon-$Greedy Q-Learning algorithm to learn the LTVs. We use the Mixed Integer Programming setup to choose the slate since the difference in computational value is minimal regarding this value of k. Along with this, it would hypothetically give us the highest return on our actions.

7. **Graphs**
   Now coming to the results - we calculate the aforementioned reward as well as the clickthrough rate as our two important measures of how well the algorithm performs and compare the Full Slate Q-Learning algorithm versus the SlateQ Decomposition.

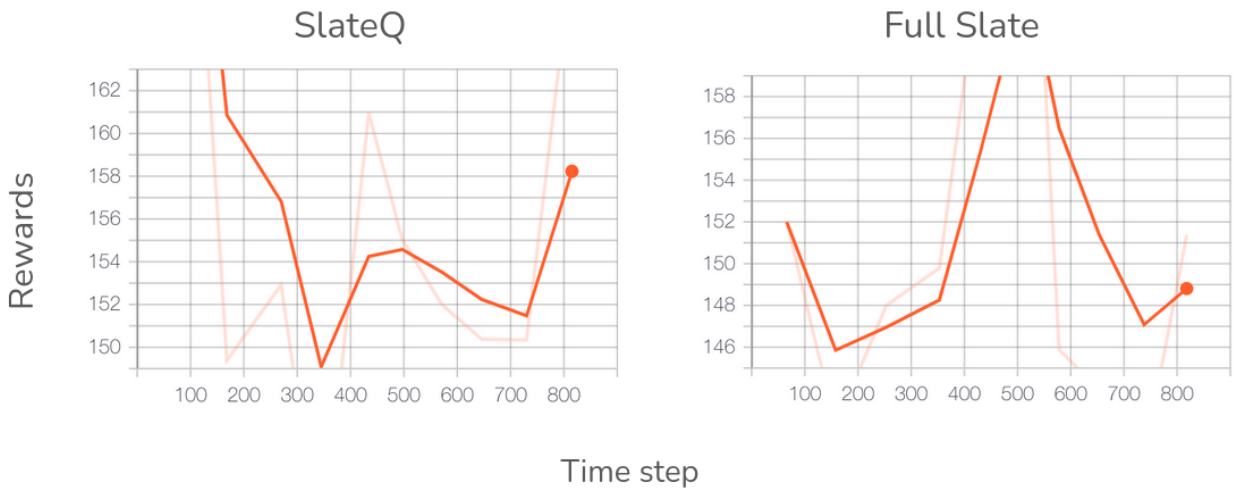


Figure 4.3: Comparing the results of the SlateQ algorithm vs the Full Slate algorithm using Average Episode Reward Ie *et al.* (2019*b*)

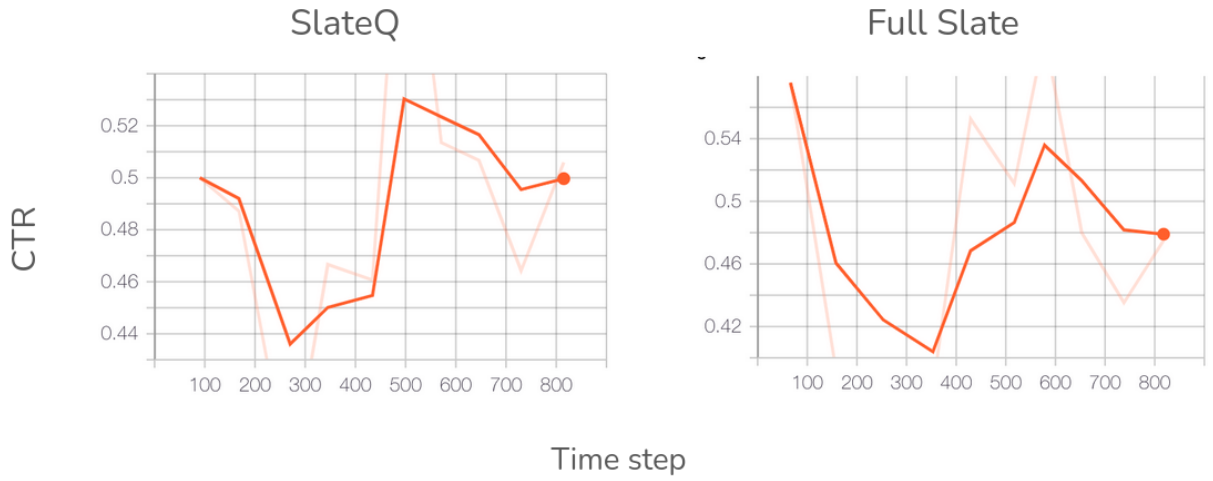


Figure 4.4: Comparing the results of the SlateQ algorithm vs the Full Slate algorithm using CTR Ie *et al.* (2019*b*)

8. **Understanding Results**

   (a) As of now, the number of actions in Slate Q would be equivalent to 10, while for Full Slate Q Learning it would be 45.

   (b) We see a significant reward increase when we move to the SlateQ decomposition Algorithm. The rewards are increasing by an average of 8 after 800 episodes of training and given the myopic nature of Full Slate Q versus calculating Long Term Values for SlateQ, there would be a widening gap if we process this further.

   (c) The reward for Slate Q Decomposition settles on **158**, while the Full SlateQ decomposition ends up at **149**. We would also expect SlateQ to outperform Full Slate Q Learning significantly as we increase the size of the corpus of documents as well as the Slate Size.

   (d) We then compare the clickthrough rate since this would also be significant for recommender systems and help determine the better algorithm. Here, we see that the CTR for Full SLateQ is **0.5** as compared to **0.48** for Full Slate Q. This indicates a **4.16**$\%$ increase which is very significant which when expanded to a setup like YouTube or Netflix, would be incredibly beneficial to the operating company.

4.3 LinUCB - Successful

1. **Philosophy**:
   Our next model uses a contextual bandit algorithm for our model's first part-Learning the slates' Q-values LinUCB is a novel idea in Recommendation Systems which is computationally efficient. Ergo, a good fit for our large data corpus can be trained on previously recorded data and adapt to constantly changing content pools. This is why we look into implementing it in this setup. The idea here is that since we use a contextual bandit, we can rank items for a certain user using LinUCB and then choose the best items. Since the rewards that we receive will affect all the items in the slate, not just the chosen one, we reckon that the best combination of items will stand out based on user features. Li *et al.* (2010)

2. **Formulation and Pipeline**:
   We have a contextual bandit i.e the bandit is based on a set of user features or context information. The algorithm takes in the current user state u in round t and the document corpus composed of $a \in \mathcal{A}$ and combines this into vector $x_{ta} \forall a \in \mathcal{A}$. Next, it chooses a set of arms $a_t \in \mathcal{A}$ and receives a payoff. This payoff is then used to improve the arm selection. This is similar to a contextual bandit. Our next step is to factor in the Upper Confidence Bound into the exploration system.
   This step should help us with a better balance between the explore exploit issue and improve on simple Bandit systems.

3. **Algorithm**: This algorithm uses a Hybrid Learning Algorithm where it assumes features that are shared by all arms along with arm-specific features.
   This adds the $z_{t,a}^T$ term to our algorithm.We use the following hybrid model:

   $$E[r_{t,a}|x_{t,a}] = z_{t,a}^\top \beta^* + x_{t,a}^\top \theta^*$$

   The first term here involves all the common features across arms, and the remaining arm-specific features are captured in the second term.
   Here $\mathbf{A}_a$ is defined as $\mathbf{D}_a^T \mathbf{D}_a + \mathbf{I}_d$ where $\mathbf{D}_a$ is a design matrix of dimension $m \times d$ at trial t, whose rows correspond to m training inputs (e.g., m contexts that are observed previously for article a), and $\mathbf{b}_a \in \mathbb{R}^m$ is the corresponding response vector (e.g., the corresponding m click/no-click user feedback).

0: Inputs: $\alpha \in \mathbb{R}_+$
1: $\mathbf{A}_0 \leftarrow \mathbf{I}_k$ (k-dimensional identity matrix)
2: $\mathbf{b}_0 \leftarrow \mathbf{0}_k$ (k-dimensional zero vector)
3: **for** $t = 1, 2, 3, \ldots, T$ **do**
4: Observe features of all arms $a \in \mathcal{A}_t$: $(\mathbf{z}_{t,a}, \mathbf{x}_{t,a}) \in \mathbb{R}^{k+d}$
5: $\hat{\boldsymbol{\beta}} \leftarrow \mathbf{A}_0^{-1}\mathbf{b}_0$
6: **for all** $a \in \mathcal{A}_t$ **do**
7: **if** a is new **then**
8: $\mathbf{A}_a \leftarrow \mathbf{I}_d$ (d-dimensional identity matrix)
9: $\mathbf{B}_a \leftarrow \mathbf{0}_{d \times k}$ (d-by-k zero matrix)
10: $\mathbf{b}_a \leftarrow \mathbf{0}_{d \times 1}$ (d-dimensional zero vector)
11: **end if**
12: $\hat{\boldsymbol{\theta}}_a \leftarrow \mathbf{A}_a^{-1}\left(\mathbf{b}_a - \mathbf{B}_a\hat{\boldsymbol{\beta}}\right)$
13: $s_{t,a} \leftarrow \mathbf{z}_{t,a}^\top \mathbf{A}_0^{-1}\mathbf{z}_{t,a} - 2\mathbf{z}_{t,a}^\top \mathbf{A}_0^{-1}\mathbf{B}_a^\top \mathbf{A}_a^{-1}\mathbf{x}_{t,a} + \mathbf{x}_{t,a}^\top \mathbf{A}_a^{-1}\mathbf{x}_{t,a} + \mathbf{x}_{t,a}^\top \mathbf{A}_a^{-1}\mathbf{B}_a\mathbf{A}_0^{-1}\mathbf{B}_a^\top \mathbf{A}_a^{-1}\mathbf{x}_{t,a}$
14: $p_{t,a} \leftarrow \mathbf{z}_{t,a}^\top\hat{\boldsymbol{\beta}} + \mathbf{x}_{t,a}^\top\hat{\boldsymbol{\theta}}_a + \alpha\sqrt{s_{t,a}}$
15: **end for**
16: Choose arm $a_t = \arg\max_{a \in \mathcal{A}_t} p_{t,a}$ with ties broken arbitrarily, and observe a real-valued payoff r_t
17: $\mathbf{A}_0 \leftarrow \mathbf{A}_0 + \mathbf{B}_{a_t}^\top \mathbf{A}_{a_t}^{-1}\mathbf{B}_{a_t}$
18: $\mathbf{b}_0 \leftarrow \mathbf{b}_0 + \mathbf{B}_{a_t}^\top \mathbf{A}_{a_t}^{-1}\mathbf{b}_{a_t}$
19: $\mathbf{A}_{a_t} \leftarrow \mathbf{A}_{a_t} + \mathbf{x}_{t,a_t}\mathbf{x}_{t,a_t}^\top$
20: $\mathbf{B}_{a_t} \leftarrow \mathbf{B}_{a_t} + \mathbf{x}_{t,a_t}\mathbf{z}_{t,a_t}^\top$
21: $\mathbf{b}_{a_t} \leftarrow \mathbf{b}_{a_t} + r_t\mathbf{x}_{t,a_t}$
22: $\mathbf{A}_0 \leftarrow \mathbf{A}_0 + \mathbf{z}_{t,a_t}\mathbf{z}_{t,a_t}^\top - \mathbf{B}_{a_t}^\top \mathbf{A}_{a_t}^{-1}\mathbf{B}_{a_t}$
23: $\mathbf{b}_0 \leftarrow \mathbf{b}_0 + r_t\mathbf{z}_{t,a_t} - \mathbf{B}_{a_t}^\top \mathbf{A}_{a_t}^{-1}\mathbf{b}_{a_t}$
24: **end for**

Figure 4.5: The Algorithm we useLi *et al.* (2010)

4. **Results Reward**: We see that the rewards here, settle at a mean of about 156,

Figure 4.6: Ploting Training Episodes vs RewardsLi *et al.* (2010)

which indicates a clear improvement over the 149 we observed with the Full Q Learning. This is also close to the SlateQ number of 158, however after more iterations which might help increase the average reward for SlateQ as well.
Learning Speed: We do see a sharp learning curve with LinUCB, however, which shows us that the algorithm was quick to learn as well.
Compute: This method was also significantly faster than the other methods in that it could complete 5000 episodes in under 10 minutes on a CPU with limited processing. While it was impossible to carry out the same process with either of the previous two algorithms, LinUCB required far less computational power and was much faster. This nudges us in the direction that as the size of the document corpus increases, and our previous two algorithms slow down, it could be beneficial to turn to LinUCB instead.

4.4 Twin Neural Network - Failed

1. **Philosophy**:
   The Twin Neural Network model is based on the idea that a non linear approach to solve the Slate Recommendation Problem might be the fastest optimal way ahead. The first step of this model incorporates neural network models to estimate the Q-values of the slates. This could be a suitable way to address the slate recommendation problem as well and this belief comes from the LinUCB model showing reasonably strong results and the NeuralUCB algorithm having worked well for simpler contextual bandits. We combine this with another neural network to solve the evaluation issue, which basically means that we have one neural network to solve the learning problem, and one to solve the evaluation problem.

2. **Agent**:
   We use a 3 layer NN, with $32 \times 32 \times 16$ nodes to understand the context between actions. We use Sigmoid activation functions with a total of 16 documents in the corpus. The algorithm takes in the document corpus and creates a feature embedding for the same. The second neural network takes in the user state u in round t and the document corpus feature embedding and combines this into vector $x_a \forall a \in A$. Next, it chooses a set of arms $a \in A$ and receives a payoff.
   The NeuralUCB algorithm uses a neural network model to estimate the Q-values of the slates, which are then used to select the best set of items for the users by the second neural network, which once again uses three layers with $16 \times 16 \times 8$ nodes. We use ReLu activation for this neural network. During the hyperparameter testing phase we modified these parameters significantly.
   We use triplet loss against the first neural network. We choose this since we want to understand how well the feature embeddings capture the ground truth.

3. **Results**:
   We faced issues with this agent. As you can see in the graph below, the agent failed to show any learning over training. It failed to do much better than a random agent and does not show signs of learning. We tried repeated iterations of the same however, we were not able to see learning in this setup.

4. **Possible Reasons for Failure**:

   (a) The most likely reason for failure could be that the neural networks were mismatched. Despite altering hyperparameters, maybe they couldn't learn from each other and that a NN might perform significantly better if -paired up with a simpler slate selection mechanism.

   (b) It could also be related to insufficient data for learning. Both the neural networks have a large number of nodes and layers, so maybe this could be a better set up if we had the computational power to train for much longer.

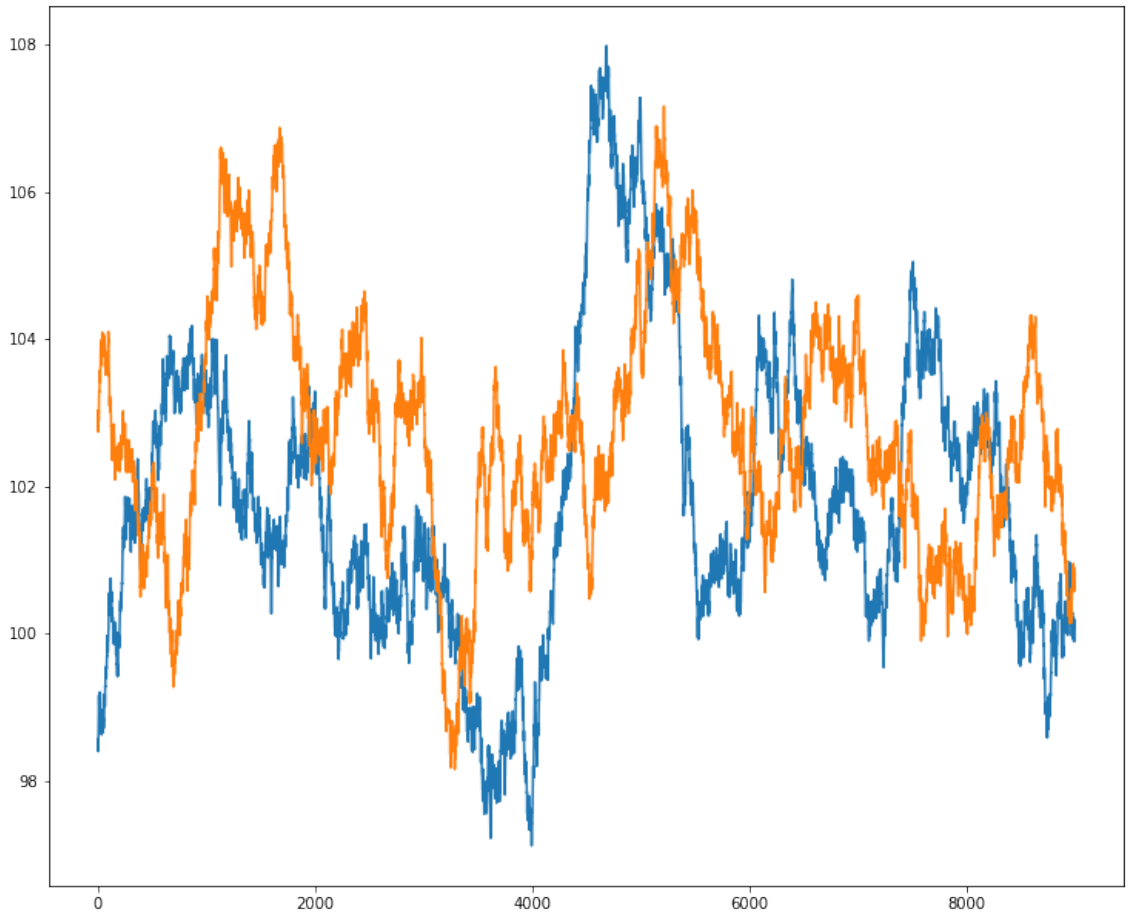


Figure 4.7: The blue curve is a random agent and the orange curve with a slightly higher average is the NeuralUCB indicating nearly 0 learning.

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this work, we have started by creating a suitable environment that allows users to test and map Slate Recommender Systems against each other. We applied traditional RL methods to establish a baseline and understood challenges faced during learning and evaluation steps in a Slate setup. We then broke the problem down into it's learning and evaluation components.

We attempted to break the first of those down using the current SoTA method - SlateQ, and a non linear method, in NeuralUCB. For the Evaluation Mechanism as well, we used 4 methods - Top K, Greedy Step, Mixed Integer Programming, and Deep Neural Network.

We also another new linear method untested in this field - Linear Contextual Bandits with Upper Confidence Bounds. Analyzing combinations of these, we saw that our best results were when **SlateQ-MIP** were paired up and with **LinUCB**. The Neural Network for evaluation also showed success with SlateQ. However the non linear method was not able to learn in this environment.

This would indicate that the item interactions that were being modelled by non linear methods did not hold as much weight as just simply selecting the strongest items in the corpus i.e a more greedy strategy that the linear methods would employ. The evaluation methods all performing similarly would then be expected from here.

The first conclusion to draw from here would be that slate recommendation problems would be best handled by using RL methods to understand the documents individually and using a separate mathematical model to combine them into slates.

The next conclusion we can derive is that while item interactions could be important in certain situations, we would need to model these based on a rule based system instead of attempting to learn them directly.

Ranking methods do show some promise however, the learning functions need to be changed if we are to adapt them directly since that could be one of the reason the DNN did not learn as much as the other methods.

Possible Future Work in the field could include -

1. Adding another layer of features in the LinUCB model to incorporate two item interactions into the model
2. Using SlateQ or LinUCB to select the top 3k items in the corpus for all users, followed by using a Neural Network to now pick the top k items from this shrunken set where it could easily factor in combinatorial interactions more easily.
3. Using alternate user choice models to change the learning problem at the first stage and then combining it with SlateQ to improve results further.

REFERENCES

1. **Chen, W.**, **Y. Wang**, and **Y. Yuan** (2014). Combinatorial multi-armed bandit and its extension to probabilistically triggered arms. *CoRR*, **abs/1407.8339**. URL http://arxiv.org/abs/1407.8339.

2. **Covington, P.**, **J. Adams**, and **E. Sargin** (2016). Deep neural networks for youtube recommendations, 191–198. URL https://doi.org/10.1145/2959100.2959190.

3. **Gao, C.**, **X. He**, **D. Gan**, **X. Chen**, **F. Feng**, **Y. Li**, **T. Chua**, and **D. Jin** (2018). Learning recommender systems from multi-behavior data. *CoRR*, **abs/1809.08161**. URL http://arxiv.org/abs/1809.08161.

4. **Glauber, R.** and **A. C. Loula** (2019). Collaborative filtering vs. content-based filtering: differences and similarities. *CoRR*, **abs/1912.08932**. URL http://arxiv.org/abs/1912.08932.

5. **Ie, E.**, **C. Hsu**, **M. Mladenov**, **V. Jain**, **S. Narvekar**, **J. Wang**, **R. Wu**, and **C. Boutilier** (2019*a*). Recsim: A configurable simulation platform for recommender systems. *CoRR*, **abs/1909.04847**. URL http://arxiv.org/abs/1909.04847.

6. **Ie, E.**, **V. Jain**, **J. Wang**, **S. Narvekar**, **R. Agarwal**, **R. Wu**, **H.-T. Cheng**, **T. Chandra**, and **C. Boutilier**, Slateq: A tractable decomposition for reinforcement learning with recommendation sets. *In Proceedings of the Twenty-eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*. Macau, China, 2019*b*. See arXiv:1905.12767 for a related and expanded paper (with additional material and authors).

7. **Joachims, T.**, Optimizing search engines using clickthrough data. *In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02. Association for Computing Machinery, New York, NY, USA, 2002. ISBN 158113567X. URL https://doi.org/10.1145/775047.775067.

8. **Kassraie, P.** and **A. Krause** (2022). Neural contextual bandits without regret.

9. **Kveton, B.**, **C. Szepesvári**, **Z. Wen**, and **A. Ashkan** (2015). Cascading bandits. *CoRR*, **abs/1502.02763**. URL http://arxiv.org/abs/1502.02763.

10. **Li, H.**, **S. Webster**, and **G. Yu** (2020). Product design under multinomial logit choices: Optimization of quality and prices in an evolving product line. *Manufacturing and Service Operations Management*, **22**(5), 1011–1025. ISSN 1523-4614. Funding Information: Funding: H. Li is funded by the W.P. Carey School of Business Dean's Award of Excellence Summer Research [Grant 2017]. Supplemental Material: The online appendix is available at https://doi.org/10.1287/msom.2019.0788. Publisher Copyright: © 2020 INFORMS.

11. **Li, L.**, **W. Chu**, **J. Langford**, and **R. E. Schapire** (2010). A contextual-bandit approach to personalized news article recommendation. *CoRR*, **abs/1003.0146**. URL http://arxiv.org/abs/1003.0146.

12. **Lin, Y.**, **Y. Liu**, **F. Lin**, **P. Wu**, **W. Zeng**, and **C. Miao** (2021). A survey on reinforcement learning for recommender systems. *CoRR*, **abs/2109.10665**. URL https://arxiv.org/abs/2109.10665.

13. **Mladenov, M.**, **C. Hsu**, **V. Jain**, **E. Ie**, **C. Colby**, **N. Mayoraz**, **H. Pham**, **D. Tran**, **I. Vendrov**, and **C. Boutilier** (2021). Recsim NG: toward principled uncertainty modeling for recommender ecosystems. *CoRR*, **abs/2103.08057**. URL https://arxiv.org/abs/2103.08057.

14. **Srinivas, N.**, **A. Krause**, **S. M. Kakade**, and **M. W. Seeger** (2009). Gaussian process bandits without regret: An experimental design approach. *CoRR*, **abs/0912.3995**. URL http://arxiv.org/abs/0912.3995.

15. **Swaminathan, A.**, **A. Krishnamurthy**, **A. Agarwal**, **M. Dudík**, **J. Langford**, **D. Jose**, and **I. Zitouni** (2016). Off-policy evaluation for slate recommendation. *CoRR*, **abs/1605.04812**. URL http://arxiv.org/abs/1605.04812.

16. **Wenga, C.**, **M. Fansi**, **S. Chabrier**, **J. Mari**, and **A. Gabillon** (2021). A comprehensive review on non-neural networks collaborative filtering recommendation systems. *CoRR*, **abs/2106.10679**. URL https://arxiv.org/abs/2106.10679.

17. **ZADL** (2019). An introduction to q-learning: reinforcement learning. URL https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/.

18. **Zhou, D.**, **L. Li**, and **Q. Gu** (2019). Neural contextual bandits with upper confidence bound-based exploration. *CoRR*, **abs/1911.04462**. URL http://arxiv.org/abs/1911.04462.