Applying Reinforcement Learning Based Tutor Strategy Recommendation Service To The ASSISTments

by

Zekun Dai

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

May 2021

APPROVED:

Professor Neil T. Heffernan, Major Thesis Advisor

Professor Craig Wills, Head of Department

Abstract

Reinforcement Learning, specifically Multi-armed bandit algorithm(MAB), has shown great results in personalized recommendation. This thesis focused on adding a reinforcement learning based personalized hints/explanations recommendation service to the ASSISTments, an online learning platform. This thesis investigated different MABs and implemented them to test on collected educational dataset. This thesis also explored the design and implementation of the infrastructure that can give support to provide tutor-strategy recommendation service for ASSISTments. This thesis conducted experiments to compare different bandit algorithms using the mean cumulative regret as the metric, Thompson Sampling among all selected was the best choice for actual production usage. By comparing both contextual and non-contextual MABs with random controlled methods for the specific application, MAB does not introduce bias as well as they do not have significant advantage over random methods.

Acknowledgements

I would like to thank my advisor, Professor Neil T. Heffernan for the guidance I received from him during my graduate study period. I would like to thank Ethan Prihar for his guidance and help over the Thesis work. His patience and attention to details has been instrumental in making me a better researcher. I would like to thank ASSITments Engineering Team: Christopher Donolley, David Magid, Joseph St. Pierre, Ryan Emberling, Tom Bolton for their support in building the infrastructure. I also want to thank my thesis reader, Professor Joseph Beck, for giving useful feedback to make this thesis a success.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Crowdsourcing for Education

Crowdsourcing by definition is a way of achieving a certain goal with the information gathered from a crowd. It is a technique that has been used in many applications including educational field. As the needs and usage of online learning, "crowdsourcing for education" (CfE) [JSB18] as a technique to develop online educational technologies receives a lot of attention from both educational researchers and developers. For the reason that crowdsourcing is aimed to deliver quality education for learners, and online education as a newly established area is relatively unexplored in terms of creating crowds[WAC+12]. Online learning technologies have provided both new methods and resources for crowdsourcing. Platform such as DALITE [BLDC16] creates a crowd of users and let them access resources and information provided by peers. PeerWise[DLRH08] crowdsources teachers and students to create and evaluate multiple choice questions, which enchances the standarad learning process.

Started since mid-1990s, learning through digital technology and online environment has become more and more popular for K-12 students and teachers. Especially

during COVID-19 times, online learning usages has increased drastically[BFT20]. As identified by many researchers, personalized learning is one of the direction of applying crowdsouring[JSB18][WAC+12].

1.2 Personalized Learning

Personalized Learning has been one of the most popular topics in educational study. The goal of Personalized Learning is to provide customized learning experience for different students for their different characteristics such as strengths and interests. A learning plan will be established at a individual level.

Personalized learning have positive effects on students learning achievement. With adoption of personalized learning, student who received personalized help improved substantially on mathematics and reading performance than their peers[PSBH15].

There are many attempts to apply personalized learning in different forms. Personalized Learning through the form of educational computer game can promote students' motivation in learning and improve their learning achievements at the same time[HSH+12]. Crowdsourcing Videos based and other learning resources is also helpful in realizing personalized learning at a large scale.[WS17].

1.3 Background

The ASSISTments[HH14] as an online educational platform founded by Professor Neil Heffernan. The ASSISTments provides teachers with various educational technologies as well as data and tools for educational researchers to examine different educational hypotheses.

Under ASSITments's Ideas Lab grant, "Just-in-time Instruction" project is dedicated to provide personalized learning to students. Since Reinforcement Learning,

specifically Multi-armed bandit algorithms(MAB), has shown great results in personalized recommendation, part of the goal is build a reinforcement learning based recommendation system.

Inspired by this, providing a personalized tutoring strategy recommendation service has became a goal for educational technology developers. TeacherAssist[PH20], as part of newly introduced services the ASSISTments. TeacherAssist allows teachers to create customized assistance messages for the specific problems they assigned to their students at the students' demand. The assistance can also redistributed and accessed by other students outside the teacher's class. With TeacherAssist service, ASSISTments now can provide different tutoring strategies for students. Students using the ASSISTments online tutor are currently given problem specific hints, explanations, and scaffolding. Hints represent a series of helpful messages that teacher created that aims to provide students with some additional information to help solve the assigned problem. When student request hint, the system will provide a single hint at once, and the student can attempt to solve the problem again, which ensures the student mastered the help material. Scaffoldings represent a set of problems that are decomposed into step-by-step guidance for students to answer the problem. Scaffolding requires students to have no prior knowledge, this also helps students who struggles to master knowlege to slove complicated problems. Explanations represent the detailed write up for answer the question along with the answer. Explanations demonstrates students the way to solve the particular problem type. As only hints and explanation are created by teachers among all three tutor strategies that are supported, this thesis will focus on these two types.

The random controlled experiments using data provided by TeacherAssist show that students has significant statistical improvement on answering the problems if they received on-demand assistance for previous problem[PH20]. With this a

foundation, the next step towards providing personalized tutoring strategy recommendation service would be building the a AI based recommendation service that can provide the best hints/explanations for each student.

1.4 Goal of the thesis

The goal of this thesis is to explore how to apply reinforcement learning in helping the system make the decision to provide students with the best tutor strategy and introduce it within ASSISTments as Reinforcement Learning Service(RLS). This involves the following tasks of both building the infrastructure and test out different reinforcement learning models, specifically multi-armed bandit algorithms(MAB).

1. To design the infrastructure that can support using multi-armed bandit algorithms(MAB) in tutor strategy recommendation decision making.

2. To investigate different multi-armed bandit algorithm models and implement these models.

3. To test out different multi-armed bandit algorithm models in same educational context and analysis each model's results.

1.5 Research Questions

This thesis intends to achieve these goals by addressing the following research questions:

1. Which Hints/Explanations are relevant to each problem?

   The effect of provided tutor strategies are indistinguishable at glance, since there's no obvious way to tell if this particular tutor strategy is relevant to this problem. Hence, to analyze if this tutor strategy is relevant, the RLS that uses MAB algorithms has to demonstrate if there is a particular tutor strategy has the good effect on students in helping them answering the particular type of questions.

2. Which Hints/Explanations is most helpful to an individual student?

   To achieve the goal of personalized learning, RLS has to be able to provide the most suitable tutor strategy to individual student, and this will be demonstrated by if the student receiving an particular tutor strategy can answer the next problem correct.

3. What features determine which Hints/Explanations is most helpful?

   The fact that individual students are different in their learning abilities, background, etc and the fact that problems are different in types, difficulties. These factors will affect the recommendation largely. This thesis will need to investigate and identify which features can determine which tutor strategy to recommend.

4. Which bandit algorithms is the most suitable for this application context?

   As there are many MAB algorithms, this thesis will need to try different MAB algortihms out and find the most suitable one in this particular application.

5. Does bandit algorithms introduce bias on the best tutor strategies comparing with random controlled experiments?

   Machine learning models have the possibility that they could incorporate bias within them. In order to find out if the MAB algorithms will introduce bias

with them, this thesis will need to compare MAB results with non-machine learning methods like using Random Controlled Experiments.

Chapter 2

Multi-armed Bandits

Multi-armed bandits algorithms(MAB) is an instance of Reinforcement Learning where in a reinforcement learning setup, agents make decisions to interact with the environment based on observations and get a feedback of the action from the environment. The term "Bandit" in "multi-armed bandits" comes from "one-armed bandit" machines used in a casino[KVJ87]. In the original set up, an agent's goal is to maximize the outcomes when playing many one-armed bandit machines that have different probabilities of win. The problem involves a trade-off between exploration/exploitation, which means the agent needs to find a balance between exploring other machines and exploiting the machine that have the best payoff so far.

The multi-armed bandit problem was formalized to is used to describe the decision-making with the outcomes to be uncertain, the outcomes can be either stochastic or adversarial, and it includes the fundamentals of reinforcement learning, such as rewards, time steps etc. In a multi-armed bandit problem, at a certain time step, there are multiple different actions/arms for the agent to choose from and the reward which is the feedback is based on the chosen action. Ideally, the

distribution of rewards of different actions is independent, and there appears to be at least one action that will give maximum reward. In this case, the goal of the agent in to identify this best arm that will produce the maximum reward after a certain number of trials. Different MABs in this case will be the agent, and the quicker and more accurate the MAB finds the best arm, the better this agent is. MABs that select action without utilizing the information from the environment(context) are referred as non-contextual multi-armed bandit algorithms, the ones does are referred as contextual multi-armed bandit algorithms.

The contextual MABs expand the non-contextual MAB decison-making by incorporating the conditional contexts. The contextual MABs can serve as a good method of penalization, but it also increase the chance of including bias in the decision making process. There are couple most popular contextual bandit algorithms: LinUCB[LCLS10] extends UCB algorithm to contextual cases. Thompson sampling[AG13] [RVRK$^+$17] has been shown to perform competitively to the state-of-the-art in a variety of bandit and other learning contexts.

Ideally, under the personalized learning scenario, the recommendation service will give hints/explanations to each student with the consideration of their different personalities. By achieving this, both non-contextual and contextual MABs will be crucial to determine what is the best option to recommend.

Problem Definition

A k-arm MAB can be formally decribed as a tuple $< A, R >$, where A is the the set of actions which represents the interaction with one slot machine and R is the reward function, which in this case, is either 0 or 1. At each time step t, among all available k arms, the agent will choose one and receive a reward r. The whole sequence over t time step would be $\{< a_1, r_1 >, ... < a_t, r_t >\}$. And the probability

distribution of the reward of the k arms is $\{\theta_1, ..., \theta_k\}$. The value function of this action a is expected reward of taking that action.

$$Q(a) = \mathbb{E}[r|a]$$

If at the time step t, the action taken is a certain arm i, the value function will be $Q(a) = \theta_i$. And the optimal action a^* has θ^* as optimal reward is:

$$\theta^* = Q(a^*) = \max_{a \in A} Q(a) = \max_{1 \leq i \leq K} \theta_i \tag{2.1}$$

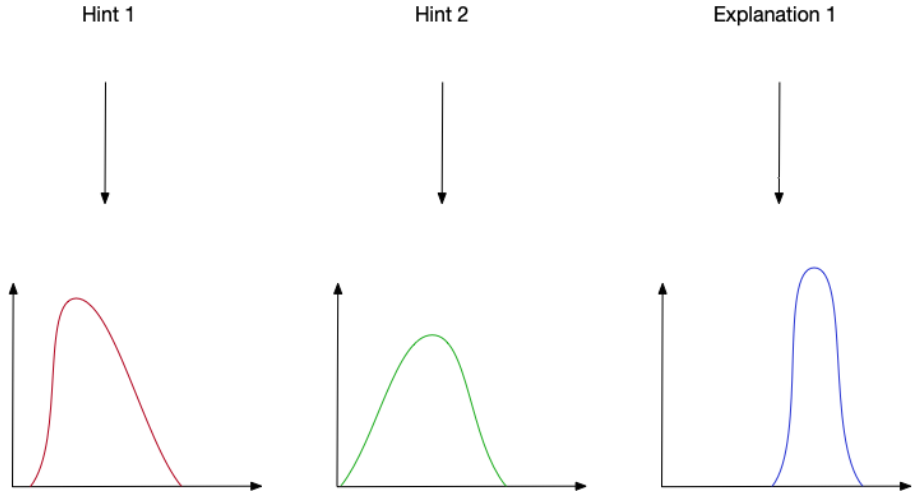
Applying to tutor strategy recommendation context

For the specific application of this thesis, the available tutor strategies(hints/explanation) are the the "arms" in this case for each problem. And each arm has a reward probability distribution, the task of the multi-armed bandits is to find the arm with the highest reward probability distribution. Take figure 2.1 as an example, the explanation 1 is the best option for that it has the highest reward distribution, if the distributions are known, it's fairly easy to make a decision. However, in actual case this are unknown to the decision make, the goal is to obtain the distribution of the following in a fast and efficient manner. That's where the multi-armed bandit algorithms mentioned comes into use.

2.1 Exploitation & Exploration

Exploitation & Exploration dilemma is one of the key factors of any decision related problems, it is one of the most basic trade-offs in nature. Exploitation & Exploration dilemma describes the trade-off between making the optimal decision by leveraging current knowledge (Exploitation) and exploring new knowledge to improve the per-

Figure 2.1: Tutor strategy reward distribution example



formance (Exploration). This is crucial for a recommendation system. For example, every time the system gets a new user, how to quickly obtain the knowledge of the user's interest in order to recommend best options for specific user. Also, if the system has the information of the best recommendation for current state, when a new recommendation option is added to current option pool, how the system will recommend the new option is another problem that need to be solved.

2.2 Regret

In a reinforcement learning setup, an agent will take actions to current environment and receive a feedback from the environment. The feedback is usually defined as a reward function that measures how good is a certain action taking by the reinforcement learning agent. The goal of the reinforcement learning algorithm is to learn the policy that will return maximum cumulative reward.

In regret theory, regret is defined as the foregone utility from the actual action choice in comparison to the case where the random states of the world are treated as known[LS82], and within regret theory, for the decision-maker is usually to minimise

cumulative regret as the optimisation problem.

Regret is a more appropriate measurement than reward as it describes the gap of not taking the optimal action.[KLM96]. Similarly to reward, regret in reinforcement learning describes opportunity loss for a certain action taken by the agent. Specifically, the difference between the optimal payoff of a possible action and the payoff of the action that has been actually taken. Formally, the optimal value is $Q(a^*)$ according to Equation 2.1. And the regret at certain step t of action taken is

$$l_t = \mathbb{E}(Q(a^*) - Q(a_t)) \tag{2.2}$$

The cumulative regret of is the sum of the regret at each step

$$L_t = \sum_{1}^{T} \mathbb{E}(Q(a^*) - Q(a_t)) \tag{2.3}$$

The goal of reinforcement learning agent now becomes minimizing the total cumulative regret.

For the specific application of this thesis, the idea is to recommend student users with tutor strategies that help them learning certain skill point. The straightforward measurement of if the tutor strategy helps is to track their performance on the next problem with the same skill point as the one they received help, the correctness of next problem. Hence, the regret at each time step t becomes:

$$l_t = \begin{cases} 1, & \text{if the student answer the next problem correctly} \\ 0, & \text{otherwise.} \end{cases} \tag{2.4}$$

2.3 ϵ-greedy

2.3.1 Standard ϵ - greedy

Greedy Algorithm does not take long-term effect of a decision into consideration and only focus on the best option available at the moment. This decision making process may pick the option that is sub-optimal and in the end does not provide maximum reward. Hence, ϵ - greedy adds randomness into the decision making process, and aims to balance exploration and exploitation[TL00]. Instead of picking the best available option every time, the algorithm introduces a ϵ parameter. For ϵ probability, the algorithm randomly explore other options instead of the current best option, and for $1 - \epsilon$ probability, it stick to current best available option.

The estimated action value is defined as:

$$Q_t(a) = \frac{1}{N_t(a)} \sum_{t=1}^{T} r_t p(a_t = a) \tag{2.5}$$

where $p(a_t = a)$ is the probability of at taking action a time step t and $N_t(a)$ is total number so far of taking action a.

And ϵ - greedy selects action a at time step t is:

$$a_t = \begin{cases} \text{action with largest } Q_t(a), & \text{with probability } 1 - \epsilon \\ \text{random select}, & \text{with probability } \epsilon \end{cases} \tag{2.6}$$

The higher ϵ value is, the more randomness will be added to the decision making process. Additionally, ϵ need to be tuned for different experiment as there is no universal best value for all setups. The algorithm's pseudocode is showed in Figure 2.2.

```
STANDARD ϵ - GREEDY (ϵ, arm_size, T)
(1) INITIALIZE(arms, regrets)
(2)     for each timestep t ∈ T
          do random generate p
             if p < ϵ
                then RANDOM SELECT AN ARM
             else p >= ϵ
                then SELECT CURRENT BEST ARM
(3)             do update arms and regrets
(4) return regrets
```

Figure 2.2: ϵ - greedy pseudocode

2.3.2 Annealing ϵ - greedy

ϵ - greedy's performance is largely rely on the choice of the ϵ value and need to be tuned according to different experiment setups. The tuning process is time-consuming and the result suffers from high uncertainty. In order to avoid setting up the epsilon values and make the algorithm parameter-free, Annealing ϵ - greedy takes idea of Simulated Annealing and sets the ϵ to decay in a constant rate with specified rule, so that it will explore less over time. Specifically, for the experiment of this thesis, the ϵ value at time step t is set to be:

$$\epsilon = \frac{1}{log(t + 1e{-}7)} \tag{2.7}$$

In such setup, at the beginning of the decision making process, ϵ value is set to be close to infinity and this would allow the algorithm to explore different options. And as time goes, ϵ value will start approaching zero and the algorithm would become more towards choosing the best available option and exploit more and more on current learnt knowledge.

The algorithm's pseudocode is showed in Figure 2.3.

```
ANNEALING ε - GREEDY (ε, arm_size, T)
(1) INITIALIZE(arms, regrets)
(2)     for each time step t ∈ T
          ε = 1/log(t + 1e−7)
          do random generate p
             if p < ε
                then RANDOM SELECT AN ARM
             else p >= ε
                then SELECT CURRENT BEST ARM
(3)             do update arms and regrets
(4) return regrets
```

Figure 2.3: Annealing ϵ - greedy pseudocode

2.4 Softmax

2.4.1 Standard Softmax

The Softmax (Boltzmann exploration) bandit is related to the Boltzmann distribution in Physics. Taking a Physics concept that systems tend to be at a more random state at high temperatures, while at low temperatures a more stable structure are formed, the algorithm sets a temperature parameter τ to affect how the agent makes its decisions.

Softmax is similar to ϵ - greedy and it enhances the effect of reward during exploration. As ϵ - greedy does not take in the information of average rewards of

different arms, Softmax sets arms with different chance of being interacted with based on the current reward average instead of uniformly exploring all arms.

At time step t, the probability of Softmax choosing a certain arm can be defined as:

$$p(a_t = a) = \frac{e^{\frac{Q_t(a)}{\tau}}}{\sum_i^A e^{\frac{Q(i)}{\tau}}} \tag{2.8}$$

When τ is large enough, the probability of exploring any specific arm is approaching 1 as the overall exponential element of all arms approach is close to 1. Otherwise, when τ is small, arms with higher average return is more likely to be chosen by the agent.

Instead of segmenting each time step to be either exploration or exploration, Softmax takes a different approach to solve the dilemma by increasing the chance of picking an arm with higher return than others and still having the options to pick arms with low return value.

Figure 2.4 shows the pseudocode of Standard Softmax.

```
SOFTMAX (τ, arm_size, T)
(1) INITIALIZE(arms, regrets)
(2)     for each time step t ∈ T
            do random select according to equation 2.8
(3)         do update arms and regrets
(4) return regrets
```

Figure 2.4: Standard Softmax pseudocode

2.4.2 Annealing Softmax

Annealing Softmax uses the same annealing process as Annealing ϵ - greedy. It decreases the temperature in a Softmax algorithm over time, specificly:

$$\tau = \frac{1}{log(t + 1e^{-7})} \tag{2.9}$$

As the τ decays over time, the agent will explore less.

Figure 2.5 shows the pseudocode of Annealing Softmax.

```
SOFTMAX (τ, arm_size, T)
(1) INITIALIZE(arms, regrets, τ)
(2)     for each time step t ∈ T
            τ = 1/log(t + 1e^-7)
            do random select according to equation 2.8
(3)         do update arms and regrets
(4) return regrets
```

Figure 2.5: Annealing Softmax pseudocode

2.5 Upper Confidence Bound(UCB)

If the number of pulling certain arm is large enough, the reward observed from the experiments will reflect the actual reward of the arm. However, in a simulated experiment setup, the sample number are often limited, which creates a cap between the actual reward and observed reward. The Upper Confidence Bound method leverages this confidence interval between the observed reward and the actual reward to help select the best option among the K arms. The confidence interval reflects

the randomness, the larger the interval is, the reward of the arm is more uncertain. And as time step increases, the agent gets more observation, and the interval will decrease. Ideally with infinite number of samples, the interval will eventually become a specific value.

To avoid inefficient exploration that the agent might explore action that is previously observed to be bad, one approach is to decrease the parameter that controls the exploration ϵ as time increases like Annealing ϵ - greedy. UCB takes another approach and is set to be optimistic and prefers the options with high uncertainty. The agent will lean towards the actions that has strong potential to have the optimal reward. And this is described by an upper confidence bound $U_t(a)$. The true reward is set as

$$Q(a) \leq Q_t(a) + U_t(a)$$

At each time step, UCB will choose the arm with the maximum upper confidence bound:

$$a_t = argmax_{a \in \mathcal{A}}(Q_t(a) + U_t(a))$$

Hoeffding's Inequality

Hoeffding's Inequality[Hoe94] is an theorem proved by Wassily Hoeffding, which provides an upper bound of a probability, the formal definition: Let $X1$, $X1$, ..., X_t be independent random variables and the value of is within the interval $[0, 1]$. Given the actual mean value $\overline{X}_t = \frac{1}{t}\sum_{i=1}^{t} X_i$, for any $u > 0$, $\mathbb{P}[\mathbb{E}[X] > \overline{X}_t + u] \leq e^{-2tu^2}$.

2.5.1 UCB1

According to Hoeffding's Inequality, let $Q(a)$ denotes the true value and $Q_t(a)$ denotes the sample value, and the upper confidence bound of the arm $U_t(a)$, the

following equation can be established:

$$\mathbb{P}[Q(a) > Q_t(a) + U_t(a)] \leq e^{-2tU_t(a)^2}$$

In order to make the choice of which arm to pull, UCB will need to compute the $U_t(a)$:

$$U_t(a) = \sqrt{\frac{-\log e^{-2tU_t(a)^2}}{2N_t(a)}}$$

UCB1 redueces the threshold $e^{-2tU_t(a)^2}$ as the more rewards observed in time and set $e^{-2tU_t(a)^2} = t^{-4}$ and take certain action at time t becomes:

$$a_t = \arg\max_{a \in \mathcal{A}}(Q_t(a) + \sqrt{\frac{2\log t}{N_t(a)}}) \tag{2.10}$$

Where $Q_t(a)$ is the mean expected reward and $N_t(a)$ is the number of choosing arm a.

Figure 2.6 shows the pseudocode of UCB1.

```
UCB1 (arm_size, T)
(1) INITIALIZE: Pull each arm once
(2)     for each time step t ∈ (K, T)
            Choose amr according to equation 2.10
(3)         do update arms and regrets
(4) return regrets
```

Figure 2.6: UCB1 pseudocode

2.5.2 UCB2

UCB2 is an adaptation implementation of UCB1. UCB2[ACBF02] lowers the upper bound of the regret, which is the gap between best reward and actual reward. Similarly to UCB1, UCB2 computes a bound for each of the K arm and then uses

the reward history and the bound value to select which arm to choose. And the difference is that UCB2 tests each arm more than once, specifically $\tau(r_j+1)-\tau(r_j)$ times. Here, $\tau(r) = \lceil(1+\alpha)^r\rceil$ and must be a integer, and α is a hyperparameter, r_i is the current time steps.

And the upper confidence bound of UCB2 sets as:

$$U_t(a) = \frac{(1+\alpha)ln(ne/\tau(r))}{2\tau(r)} \tag{2.11}$$

Figure 2.7 shows the pseudocode of UCB2.

```
UCB2 (α, arm_size, T)
(1) INITIALIZE: Pull each arm once
(2)     for each time step t ∈ (K, T)
            for times in τ(r_j + 1) - τ(r_j)
                Choose arm according to equation 2.10
(3)         do update arms and regrets
(4) return regrets
```

Figure 2.7: UCB2 pseudocode

2.5.3 LinUCB

All the bandit algorithms above are based on the observation of the expected return, however, they does not consider the how different each user is. Hence, they are all context-free.

LinUCB was first introduced in recommending news paper articles[LCLS10]. LinUCB takes in the users' interests as one of the factors that affect the recommending strategy. It takes in different features such as interests, preferences, region,

etc of an individual user and compose those features as contexts. LinUCB believes that the context and the final return has a linear relationship.

In the particular application, the context of user features are introduced in feature creation section which answered RQ3. Those features are designed specificity to reflect the differences of each individual student.

Formally, the contexts are introduced as a d dimensional feature vector $x_{t,a}$, and the agent will now not only observe the return but also the contexts. And based on the action $a_t \in A_t$ taken, the observed payoff will be r_{t,a_t}. And for T time steps, it will observe $x_{a,t}, a_t, r_{t_{at}}$. And the expected reward of an arm is believed to have linear relation in vector $x_{t,a}$ with a certain coefficient vector θ, specifically:

$$E[r_{t,a}|x_{t,a}] = x_{t,a}^T \theta_a \tag{2.12}$$

And based on the expectation of reward, the recommendation strategy for each time step t is:

$$a_t = argmax(x_{t,a}^T \theta_a + \alpha\sqrt{x_{t,a}^T A_a^- 1 x_{t,a}}) \tag{2.13}$$

Figure 2.8 shows the pseudocode of LinUCB.

2.6 Thompson Sampling

Above methods takes Frequentist Inference approach, which set each arm with a specific value and use the values as criteria to compare different arms. And if have

LinUCB (arm_size, T)
(1) **for** each time step $t \in T$
Observe features of all arms $a \in A_t : x_{t,a}$
for $a \in A_t$ **do**
if a is new **then**
$A_a \leftarrow I_d$(d-dimensional identity matrix)
$b_a \leftarrow 0_d$(d-dimensional zero vector)
$\theta_a \leftarrow A_a^- 1 b_a$
$_{t,a} \leftarrow \theta_a^T x_{t,a} + \alpha\sqrt{x_{t,a}^T A_a^- 1 x_{t,a}}$
Choose arm $a_t = argmax(p_t)$
(3) **do** update arms A and $regrets$
(4) **return** $regrets$

Figure 2.8: LinUCB pseudocode

infinite samples to experiment, the value will approach the actual reward. Since the samples are limited in actual case, the value is approximated and use a bound to measure the gap between estimated and actual value. Thompson Sampling, on the other hand, takes Bayesian Inference approach and set the criteria to follow a probability distribution. The idea is to repeat the process of using observed history to update the likelihood function of this probability distribution to get the posterior probability. As the samples being observed increases, the density of this probability distribution will be more closer to the actual probability distribution.

At each time step t, Thompson Sampling will select the optimal action a according to the probability:

$$\begin{aligned} p(a|h_t) &= \mathbb{P}[Q(a) > Q(a^{'}, \forall a^{'} \neq a|h_t)] \\ &= \mathbb{E}_{R|h_t}[p(a = arg\max_{a \in A} Q(a)] \end{aligned} \tag{2.14}$$

where $p(a|h_t)$ is the probability of taking a certain action under current played

history h_t.

In a Bernoulli bandit setup, the action value $Q(a)$ is set to follow the Beta distribution, hence the value of $Beta(\alpha, \beta)$ is within the interval [0, 1], The two parameter α and β counts if the action is good or bad with the reward being 1 or 0. For K arms, the parameter sets $\alpha = [\alpha_1, \alpha_1, ..., \alpha_k]$ and $\beta = [\beta_1, \beta_1, ..., \beta_k]$. At each time t, the expected reward of each action $Q(a)$ will be sampled using the prior distribution $Beta(\alpha_i, \beta_i)$. And the agent will select the best action:

$$a_t = argmax(\theta_i, \frac{\alpha_{t,i}}{\alpha_{t,i} + \beta t, i}) \tag{2.15}$$

With the action, the observed reward will be computed and used to update the the Beta distribution accordingly by using the already known prior probability to compute the posterior.

$$Beta(\alpha_k, \beta_k) = \begin{cases} Beta(\alpha_k, \beta_k) + (r_t, 1 - r_t), & \text{if } a = a_k \\ Beta(\alpha_k, \beta_k), & \text{otherwise.} \end{cases} \tag{2.16}$$

Figure 2.9 shows the pseudocode of Bernoulli Thompson Sampling, the alphas and betas are the parameters for beta distribution, the successes and failures are counting the times of getting a reward 1 and 0.

THOMPSON SAMPLING $(alphas, betas, successes, failures, arm\_size)$
(1) INITIALIZE$(alphas, betas, successes, failures)$
(2)**for** each time step $t \in T$
(3) **for** each arm $i \in 1, ..., K$
do SAMPLE $\theta_i \sim Beta(\alpha_t, i, \beta_t, i)$
do SELECT ARM $A_t = argmax(\theta_i, \frac{\alpha_{t,i}}{\alpha_{t,i}+\beta t,i})$
do UPDATE $Beta(\alpha_t, i, \beta_t, i)$ ACCORDING TO 2.16 AND $regrets$
(5) **return** $regrets$

Figure 2.9: Thompson Sampling pseudocode

Chapter 3

Infrastructure design

Building infrastructure that supports providing Reinforcement Learning based tutor strategies to students on ASSISTments is one of the goal of this thesis. The infrastructure will be seamlessly integrated into ASSISTments as an additional service and communicate with different components within ASSISTments currently. Also the infrastructure will not only serve as a new feature of ASSISTments but also help better design and train different bandit models and support other educational researches.

3.1 Requirements of the system

1. Functional requirements:

    1. When Student requests help with a specific problem, the system will generate student-specific tutor strategy recommendations.
    2. When tutor strategy recommendations are not available, the system can random generate recommendations.
    3. The system can update the model in online fashion.

4. The system can trace the if the recommendation is helpful or not by automatically lookup the next problem correctness.
5. The system can calculate the different features daily.

2. Non-functional requirements:

    1. The system can support different kind of tutor strategy recommendations, and is extendable when new types of tutor strategy are developed.
    2. The system is seamlessly integrated with different components within current ASSISTments ecosystem.
    3. The system supports the feature extension.
    4. The system supports the model extension.

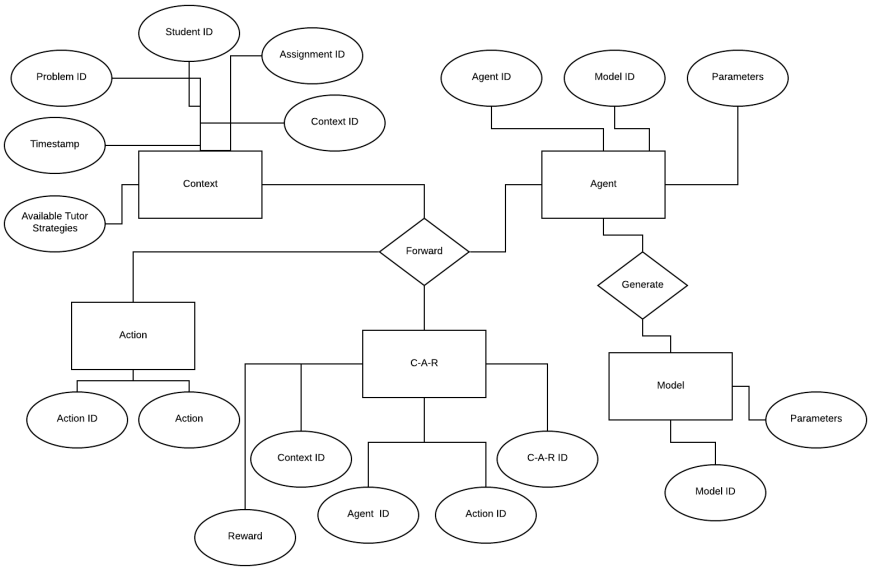
3.2 Techstack

ASSISTments platform built its whole ecosystem with Spring, a Java application framework and inversion of control container for the Java platform. In order to integrate the reinforcement learning based recommendation service into ASSISTments seamlessly, the infrastructure is written in Java using ASSISTments SDK2.0. The project is created as a JEE web project. As described in requirements, the service will intact with different components in ASSISTments, specifically the tutor, teacherAssist, and ASSISTments database. The connection with tutor to display the recommendation is specified in a Java Manifest file format. The communication with teacherAssist is done with wrapping the methods into a jar file format. The communication with ASSISTments database is using spring-jdbc. Lastly, with the nightly update task, it is done with spring scheduler and have the task running as a service on a Tomcat server.

3.3 Database design

Figure 3.1 shows the database design of RLS. RLS stores different context table for model training. The system stores different model in the model table.

Figure 3.1: ER Diagram of Reinforcement Learning Service



And the class diagram of RLS is listed in the appendix as A.1.

3.4 Workflow

General workflow

ASSITments as a whole is a well-designed platform that involves a lot of different components and services. In order to integrate the Reinforcement Learning Service seamlessly into the platform, the implementation will utilize the ASSITments SDK2.0 and communicate with the existing TeacherAssist service. The very first step will be designing the features for the reinforcement learning model, and then TeacherAssist will collect the available tutoring and provide a list of options to the

RLS. The RLS will rank the options with the models and return a ranked list to TeacherAssist. TeacherAssist will use the ranked list to decide which tutoring to provide to the student The RLS will record the request in the database.

Every night, the RLS will look through the recorded requests and determine the action taken and reward received, then update each model with the new data every night, the student and problem features will be recalculated. Figure 1 shows the workflow of the whole process of the service.

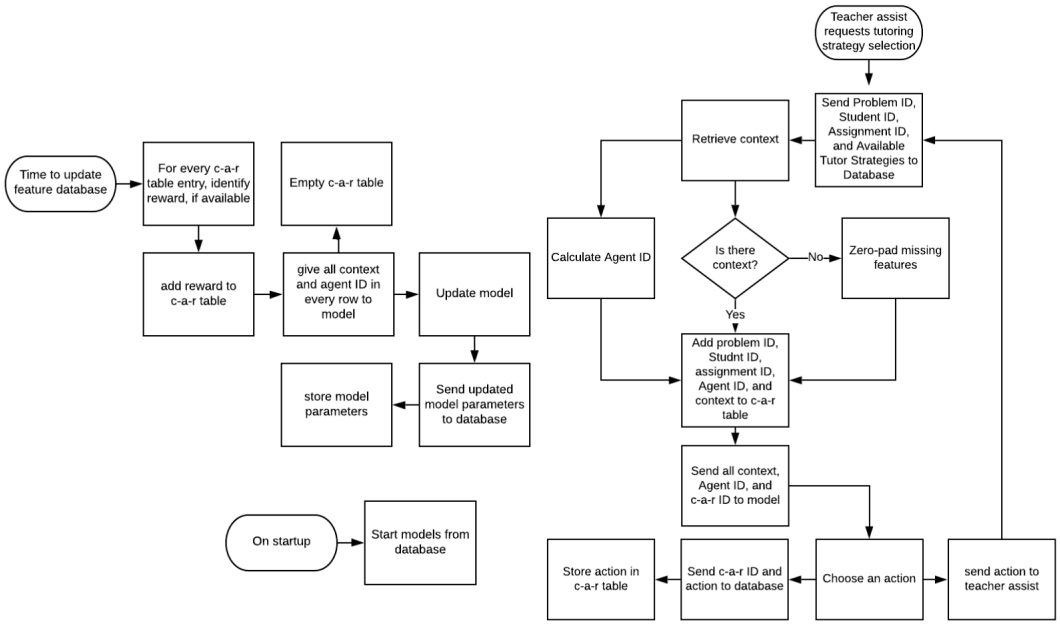


Figure 3.2: Reinforcement Learning Flowchart

Reward calculation process

Specifically, for the reinforcement learning, finding the reward is an important part of the infrastructure, and it's going to be application related reward. For specific Reinforcement Learning applications, for example, the game-playing agents, the reward is the final score of the score. As the bandit algorithms used here is for re commanding tutor strategies, the best criteria would be the score of a certain problem that

the students achieved. Figure 3.3 shows the workflow of reward calculation process. First, the service gets a request from TeacherAssist, then using the the information in the request to query the database to find where exactly did the student request the help, this step marks the start of the finding reward process. Then the service will check if the problem is answered before the students actually watched through the hints/explanations. If the answer is no, which means the student watched the tutor strategy before answer the problem, the tutor strategy has an effect on the student. If it's good tutor strategy for the student, the score would be positive and if it's not, the score will also reflect that is this tutor strategy suitable for this specific student. If the answer to the previous question is yes, then this problem's score cannot be counted as a reward since the tutor strategy doesn't affect the student's answer. Then the service will locate the next problem with the same skill code, to ensure that it's the tutor strategy has an effect on student's answer, then the score of this problem will be count as the reward.

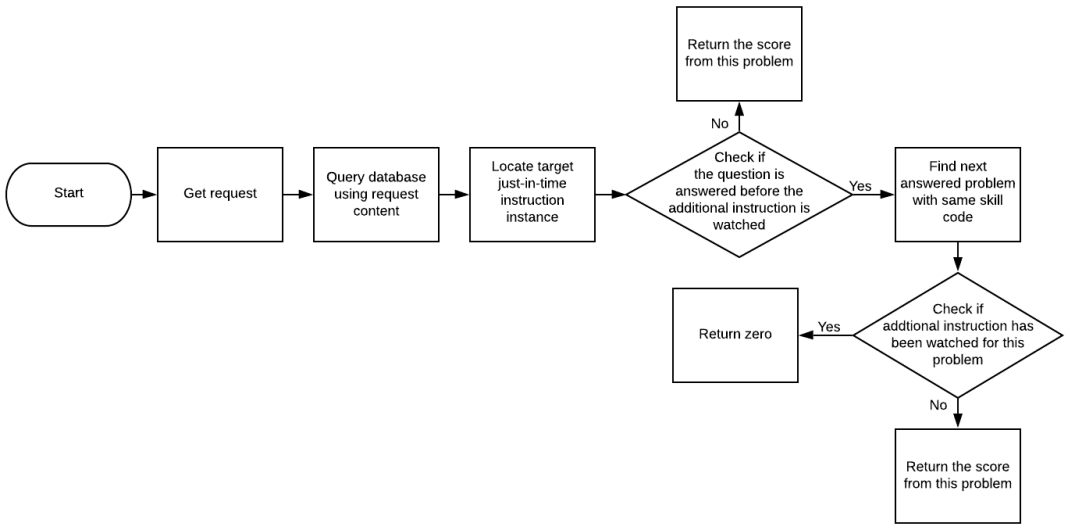


Figure 3.3: Reward Calculation Flowchart

Feature Creation

The bandit algorithm increases false positive rates when used to conduct educational experiments [RYW19]. To mitigate this effect, features that are likely to correlate with learning are being created. This method for selecting features led to the creation of a small number of features to be used as context by the multi-armed bandit algorithm.

Three problem features is chosen. These features do not include the skill associated with the problem, as that feature is used to determine which multi-armed bandit instance to use to choose tutor strategies for that problem. The first of the three features is a one-hot-encoded, categorical representation of the "type of problem", e.g., multiple choice or short answer. This feature is included because studies [FSA+18] [But18] have shown that students learn different amounts from different types of problems. The other two features are based on studies [SDW+18] that built models that incorporated problem difficulty to improve student learning. "the median percent correct" and "the median time to complete each problem" are used as an analog for problem difficulty.

The features created for students are also motivated by a desire to ensure the multi-armed bandit algorithm acts fairly. While there are many definitions of algorithmic fairness, in the context of education, fairness has to do with equity, not equality. Unlike many of the fairness definitions in literature, which strive to give every group the same content, the definition used to give every group the best content for that group. This is motivated by an understanding that marginalized groups may not have been given the same opportunities as the non-marginalized groups, which may necessitate additional tutoring or a different tutoring strategy for those groups. Striving to provide each group with content most effective for that group works toward closing the achievement gap. And the specific features are:

"Median time after clicking additional instruction before interacting with the tutor over past 50 problems", "Percent of tutoring used on past 50 problems", "Percent of secondary tutoring used on past 50 problems", "Percent of problems correct on past 50 problems", "Median time to answer a problem on past 50 problems", "Gender", "Race"

In order to measure the fairness of the multi-armed bandit algorithm, gender and race features. These features are based on lookup tables created from online repositories of names and the United States census data.

Nightly update process

Every night, the RLS will collect the problem logs and assignment logs that happened within the 24 hour time frame for feature update. This process aims to provide the updated features for retraining the different bandit algorithm models. Figure 3.4 shows the process of nightly update process.

For student features, RLS will query through the ASSISTments database for the assignment logs of the past 24 hours and then calculate the sutdent features described in the section above and the same for problem features.

3.5 System Components

3.5.1 DAO layer

DAO Layer is designed to communicate with the database, as showed in Appendix A.1. DAO classes included in RLS are ProblemFeatureDao, StudentFeatureDao, AdditionalInstructionDao, RLSModelDao, BanditFeatureDao, RLSLogDao.

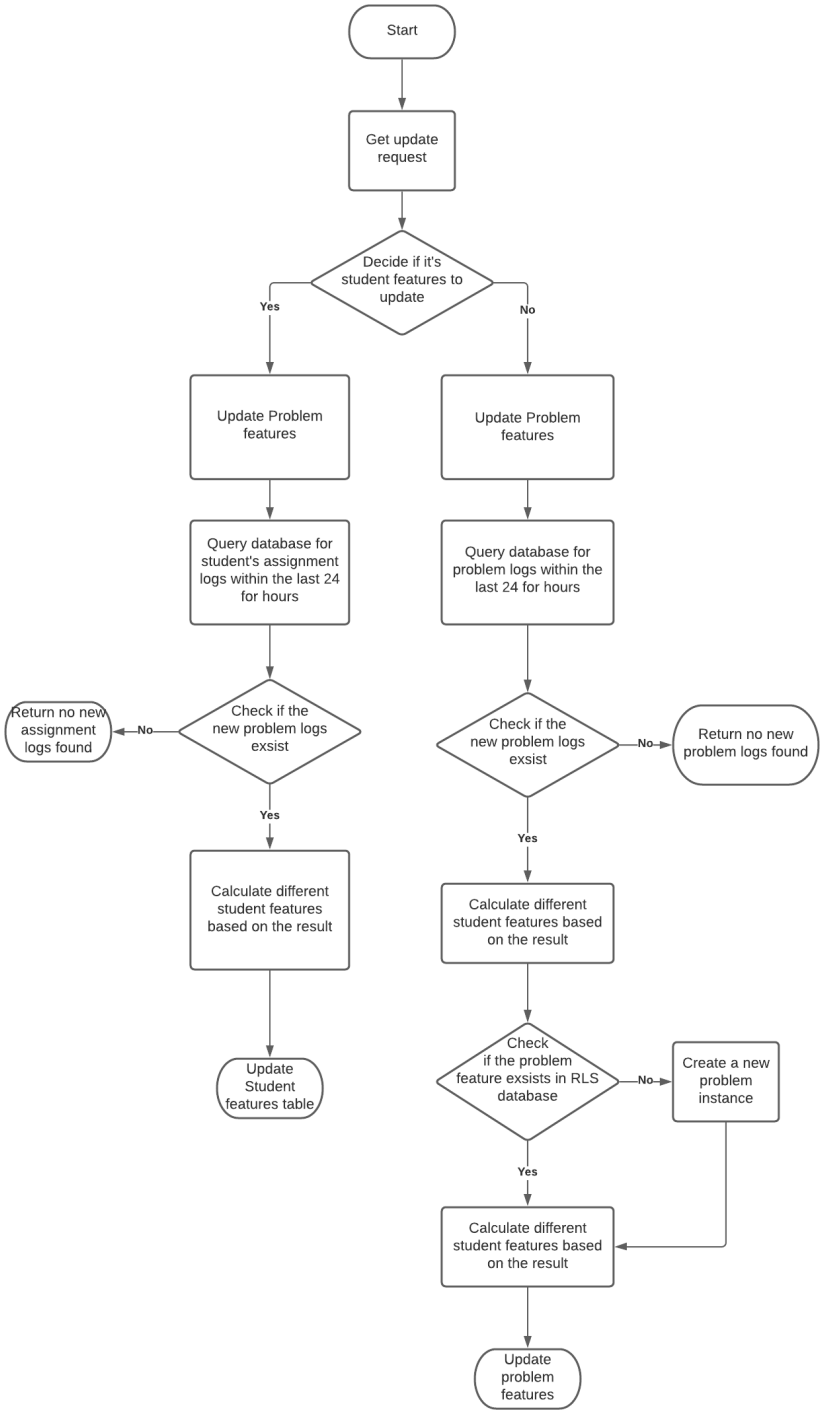


Figure 3.4: Nightly Update Flowchart

ProblemFeatureDao

Variable name	Type
id	int
problem_id	int
problem_type_id	int
mean_discrete_score	double
median_time_to_complete	double
is_active_feature_row	boolean
calculation_timestamp	Timestamp

StudentFeatureDao

Variable name	Type
id	int
student_xref	String
mean_discrete_score	double
class_percentile	double
mean_number_of_attempts	double
mean_number_of_attempts_with_hints	double
percent_help_requested_before_attempt	double
median_problem_completion$_time$	double
median_time_between_responses	double
median_time_between_hint_requested_and_interaction	double
median_time_spent_without_attempting_before_hint	double
gender	String
ethnicity	String
is_active_feature_row	boolean
calculation_timestamp	timestamp

AdditionalInstructionDao

Variable name	Type
id	integer
skill_id	integer
value	string
type_id	integer

RLSModelDao

Variable name	Type
id	integer
model_id	integer
model_type	String
model_values	String
model_keys	String

RLSLogDao

Variable name	Type
id	integer
problemfeature$_id$	integer
studentfeature_id	integer
tutorstrategy_type_id	integer
turtorstrategyfeature_id	integer
predict_reward	double
recommendation$_t imestamp$	Timestamp
next_problem_correctness	boolean
next_problem_correctness_timestamp(	Timestamp

3.5.2 Manager Layer

In Spring framework, manager layer serves as the program's logic control, it follows the Manger Design Pattern. Manager classes

RequestManager

The Request Manager builds this RequestInfo by gathering the student, problem, and tutor strategy context objects provided by the Context Manager. Request Manager is set to be able to make multiple predictions for different tutor strategy types.

- Method name: selectTutorStrategy

  Parameters: String studentXref, int problemID, int problemID, HashMap<String, ArrayList<Integer>> tutorStrategies

  Return: HashMap<String, Integer>

ModelManager

The Model Manager is in charge of the tutor strategy recommendation. The purpose of the loadModel function is to take the context from RequestInfo and decide which model to choose. (Currently, it is based on whether it is a Tutor Strategy or Additional Instruction) The purpose of the formContextArray function is to form an array of doubles that the chosen model will understand and use to make a predicted reward. When the model is loaded and the context array is formed, the ID of the model and the predicted reward from the model are put in the ModelInfo object which is returned to the Request Manager.

- Method name: getStudentContext

  Parameters: String studentID

  Return: StudentContext

- Method name: getProblemContext

  Parameters: int problemID

  Return: ProblemContext

- Method name: getTutorStrategyContext

  Parameters: String tutorStrategyType, int tutorStrategyID

  Return: TutorStrategyContext

LogManager

LogManager is created to handle all the log information. An RLS log entry is created when a tutor strategy is assigned. LogManager uses the RLSLogDao to update the information in the database.

- Method name: writeLog

  Parameters: RequestInfo requestInfo, ModelInfo modelInfo

  Return: boolean

FindRewardManager

FindRewardManager is created to find the reward of a certain tutorstrategy given by RLS. The process is described in the previous workflow section.

- Method name: getTargetProblemActions

  Parameters: XInfo userID, XInfo assignmentID, int problemID

  Return: List<Action>

- Method name: additionalInstructionTracker

  Parameters: List<Action> allActions

  Return: boolean

- Method name: findNextProblemWithSameSkillCode

  Parameters: XInfo userID, XInfo assignmentID, int problemID

  Return: int

- Method name: calculateReward

  Parameters: XInfo userID, XInfo assignmentID, int problemID

  Return: int

UpdateManager

This class handles the nightly updating of Problem and Student features. It contains one public method "updateFeatures" which, when called, will update the Problem and Student features based on the previous day's data, and add new entries to the ProblemFeatures and StudentFeatures tables.

- Method name: updateFeatures

  Parameters: None

  Return: None

Chapter 4

Experiments

4.1 Comparing different bandit algorithms on the ability of recommending different tutor strategies

4.1.1 Dataset

The thesis conducts the experiments on the data from the ASSISTments. After TeacherASSIST's deployment for 3 years, 40,292 instances of on-demand assistance were created for 25,957 distinct problems across different curricula. The timestamp for the data is by the end 2029-2020 school year.

4.1.2 Preprocessing

In order to test different bandit algorithms, the thesis defines treat the Hint/Explanation as a specific arm, and the students answer the next problem correctness as the regret. Hint/Explanation that has been used over 50 times were selected to ensure

the bandit algorithm has enough chance to explore different arms. Also, the available tutor strategy(arm) number is at least 2 to allow the bandits to make decision. After the prepossessing, the data contains 104 different questions and 240 different tutor strategies. A total number of 28,700 instances were selected with distinct Hint/Explanation. The figure below shows an example of the data sample.

The table 4.1 shows data samples of the processed data.

id	teacher_id	student_id	tutor_strategy_id	tutor_strategy_type	problem_id	next_problem_id	next_problem_correct
6337	488160	523294	Hint	1208538	1501983	1502001	0
23169	485865	523425	Explanation	1180174	1501233	1501234	1
23170	436919	523447	Explanation	1183056	1501233	1501234	1
...	...	...	...	...	...	...	...
23203	436919	523648	Explanation	1183056	1501233	1501234	1
23205	436919	523654	Explanation	1183056	1501233	1501234	1
34883	485865	490980	Explanation	1180463	1501642	1501644	1
34884	485865	490980	Explanation	1181094	1501644	1501645	1

Table 4.1: Processed data samples

teacher_id represents the teacher that created this specific tutor strategy. student_id represents the student that requested this help. tutor_strategy_id and tutor_strategy_type denotes the specific tutor strategy and its type(Hint/Explanation, excluding Scaffolding). problem_id marks which problem the student asked for help. next_problem_id is to track after the help which problem the student did, and next_problem_correct tracks if the student did correct or not after watching the tutor strategy.

4.1.3 Methodology

To simulate a reinforcement learning setup and test out different bandit algorithms, the idea is to for each problem of the total 104 problems, the bandit will random sample different samples and compute the overall mean accumulative regret as a metric to compare which bandit algorithm has the best performance in this specific application context. In order to rule out the randomness effect, each algorithm will

be executed 100 times and the mean cumulative reward is computed for each bandit algorithm. In each of the 100 trials, the time step should be significant larger than the arm number to allow the bandit algorithms to explore every arm at least once, the time step is set to be 500. At each time step, the agent will make an action and compute the regret of this choice.

4.1.4 Results

4.1.5 ϵ -greedy

Standard ϵ -greedy

For Standard ϵ - greedy, the key parameter to determine the bandit algorithm to explore new action or taking the current best action is ϵ. For ϵ probability, the agent will take a random action, otherwise for $1 - \epsilon$ probability, the agent pick the best action that it has learnt so far.

To satisfy the purpose of using bandit to recommend service in a production environment, the agent is expected to be able to be able to rank the available tutor strategies quickly and efficiently. And since at the very beginning of the deployment of the data samples are very small, the agent might not be able to find the universal best option. Hence, in order to avoid this kind of cold start, the regret is set to be the metric. The idea scenario would be the agent can quickly find the an above average tutor strategy and stick with it so that most of the students would get a fairly good one, and at the mean time the agent still needs to consider other available tutor strategies as well.

In order to learn the best ϵ that satisfies the purpose, the experiment set ϵ in a set of values and run the proposed simulation with each value and choose the one with the best result to represent standard ϵ - greedy algorithm and compare with

other bandit algorithms.

The table 4.3 demonstrates the mean cumulative regret of using standard ϵ - greedy with different ϵ.

ϵ	mean cumulative regret
0.01	93.368
0.1	55.218
0.2	63.669
0.3	73.867
0.4	84.654
0.5	95.778
0.8	129.257

Table 4.2: mean cumulative regret of using standard ϵ - greedy with different ϵ

In extreme cases, when $\epsilon = 0.01$, the agent the agent sticks to the 'best' arm too early with the low sample number that this 'best' might not be the case. When $\epsilon = 0.8$, the agent hardly exploits, it explores most of the time. Both situations, the performances are not ideal. Among all different ϵ value, $\epsilon = 0.1$ has the best performance in terms of the regret. However, when the ϵ is set to a large value, the bandit will converge faster as it explores more to find the best arm.

The 4.1 shows the regret plot over 100 trials. The regret is sparse in terms of different trials, the algorithm learns the best action to take in different times.

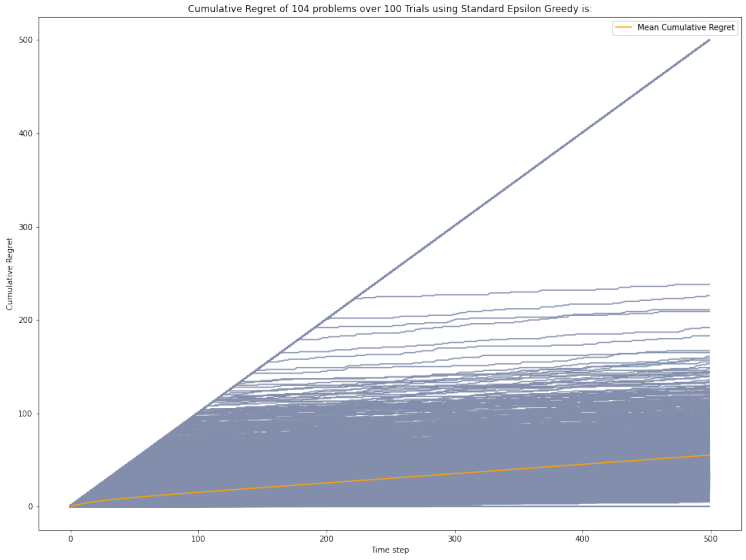
Annealing ϵ -greedy

While Standard ϵ - greedy depends on different ϵ to explore different options, Annealing ϵ -greedy is more of a straight forward and parameter free implementation. The algorithm sets the decaying epsilon with time and runs with no hyper-parameter configurations. As described in the previous section, the annealing process will set the epsilon to follow:

$$\epsilon = \frac{1}{log(t + 1e{-}7)} \tag{4.1}$$

Ideally, the, at the beginning of the decision making process, ϵ value is set to be

Figure 4.1: Cumulative regret of Standard ϵ-greedy over 100 trials



close to infinity and this would allow the algorithm to explore different options. And as time goes, ϵ value will start approaching zero and the algorithm would become more towards choosing the best available option and exploit more and more on current learnt knowledge.

The 4.2 shows the regret plot over 100 trials. The regret is less sparse compares to standard epsilon, the figure indicate that generally, the bandit started to converge around cumulative regret of 60.

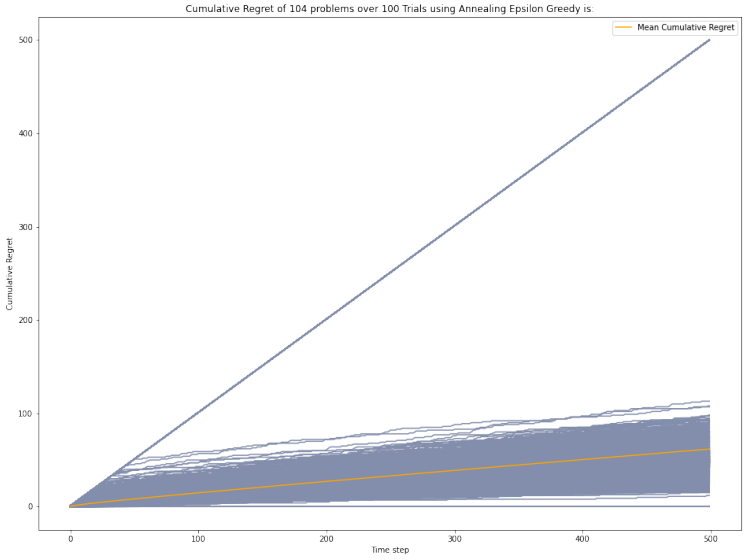
4.1.6 Softmax

Standard Softmax

The Softmax algorithm uses the reward information rate of different arms to balance the explorationexploitation.

The 4.3 shows the regret plot over 100 trials.

Figure 4.2: Cumulative regret of Annealing ϵ-greedy over 100 trials



τ	mean cumulative regret
0.01	38.958
0.03	38.972
0.05	38.982
0.1	38.995
0.2	40.682
0.3	47.546

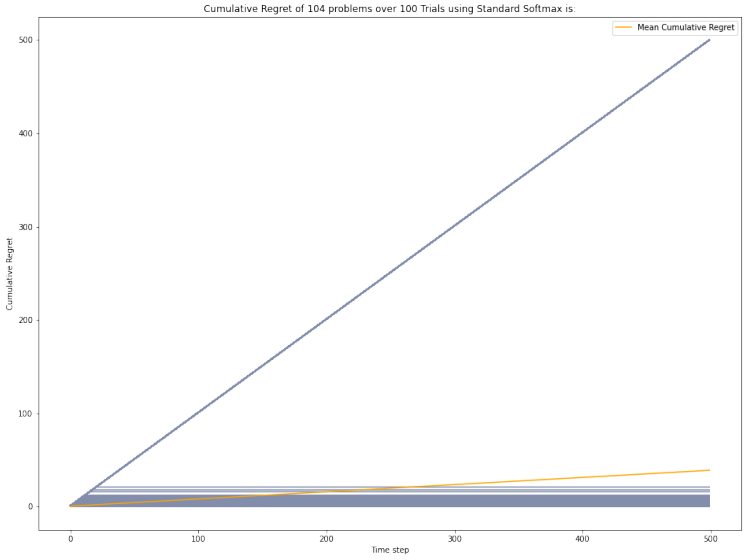
Table 4.3: mean cumulative regret of using standard Softmax with different τ

Annealing Softmax

Annealing softmax decreases the temperature in a Softmax algorithm as time increases, which means the there will be less exploration for the agent over time.

The 4.4 shows the regret plot over 100 trials. Annealing Softmax is slightly stable than Standard Softmax, and the mean cumulative regrets is 41.479 which is slightly worse than Standard Softmax.

Figure 4.3: Cumulative regret of Standard Softmax over 100 trials



4.1.7 Upper Confidence Bound(UCB)

UCB1

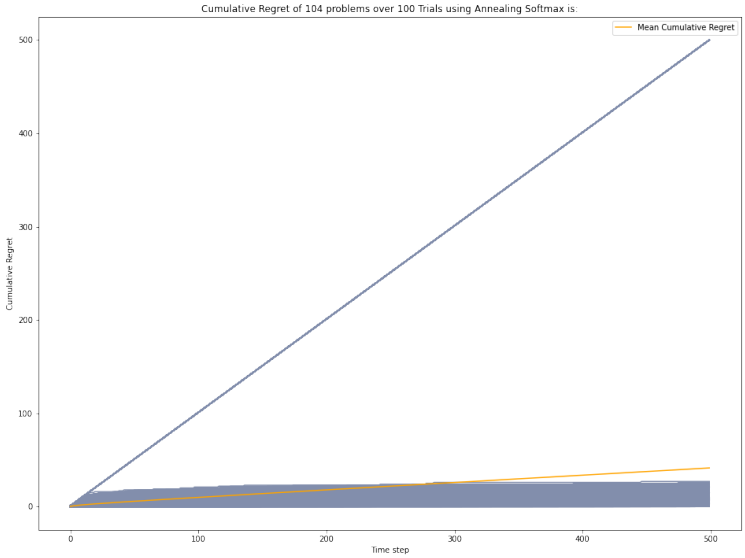
UCB1 has a significant advantage over $\epsilon - greedy$ bandits, the fact that the regret plot 4.5 is tighten within a very small range.

UCB2

As long as the α is kept relatively small, the parameter will be relatively ineffective for UCB2[ACBF02]. Hence, the α is set to 0.001 as recommended in the paper.

Figure 4.6 shows the result of UCB2. Comparing with UCB1, UCB2 is stable in terms of the mean cumulative regret. Also UCB2 can find the best arm faster than UCB1.

Figure 4.4: Cumulative regret of Annealing Softmax over 100 trials



4.1.8 Thompson Sampling

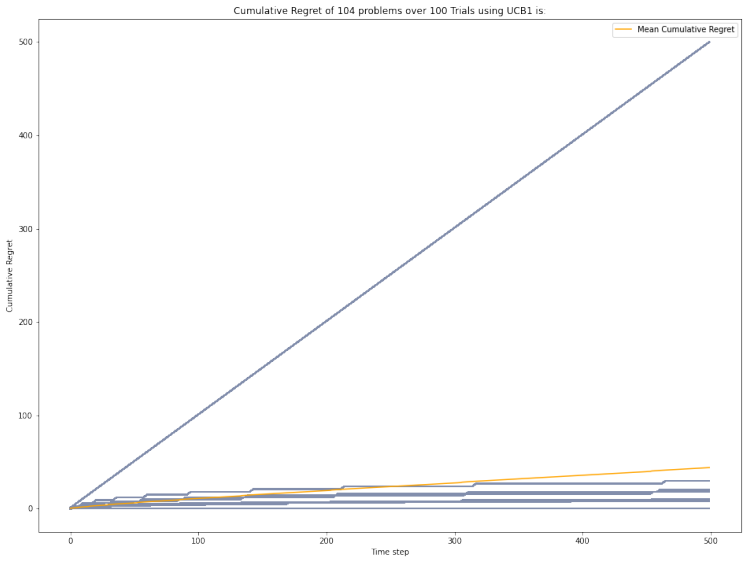
Thompson Sampling is relatively stable through the experiments. As Thompson Sampling does not require hyperparameter tuning, it can quickly find the best arm to pull.

4.1.9 Analysis

Figure 4.8 and Figure 4.9 show the parallel comparison of different bandit algorithms, the figure not only shows the mean cumulative regrets, but also the range of regrets value of the 100 trials. The range is included as a metric to demonstrate how stable the bandit algorithm is, as in 4.8.

For example, although Standard epsilon greedy has the highest mean regret, its best performance and worst performance are closer than the performances of UCB1,

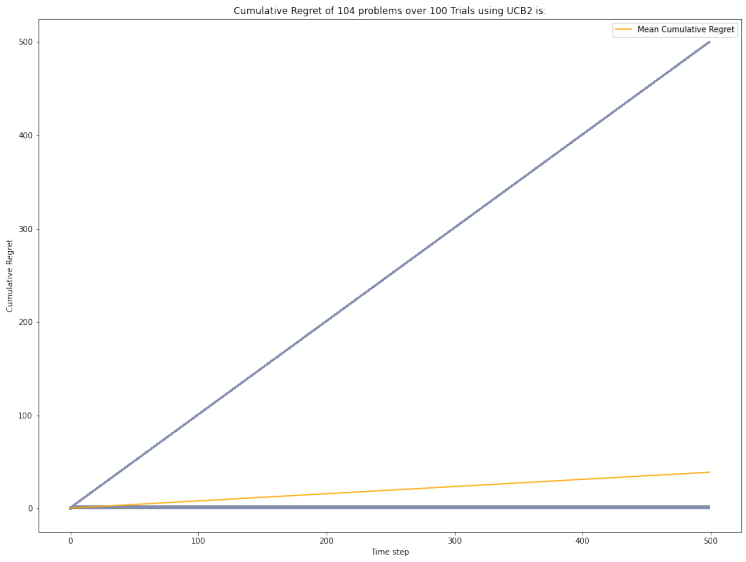
Figure 4.5: Cumulative regret of UCB1 over 100 trials



hence its more stable than UCB1. Even though UCB1 has a better overall result than Standard epsilon greedy, if were to put into actually production usage, the might produce unwanted result.

Among all the bandits, Standard Softmax, UCB2, Thompson Sampling are the ones have best results, and they are relatively close in performance and stability. However, Standard Softmax UCB2 does require hyperparameter tuning, which is more troublesome and costing if were to put into production usage. Also, UCB methods use empirical mean as the base of deciding which action to take, it does not take the arm's reward distribution into consideration, which mean if the arms are good enough, the result might be good, but if arms are not, the results might not be that ideal. In general, Thompson Sampling considers the reward distribution of different arm, and use the probability distribution as the base of selecting different action, it is not affected by how good or bad the arms are. Another advantage of Thompson Sampling is that it does not require hyperparamer tuning which made it

Figure 4.6: Cumulative regret of UCB2 over 100 trials



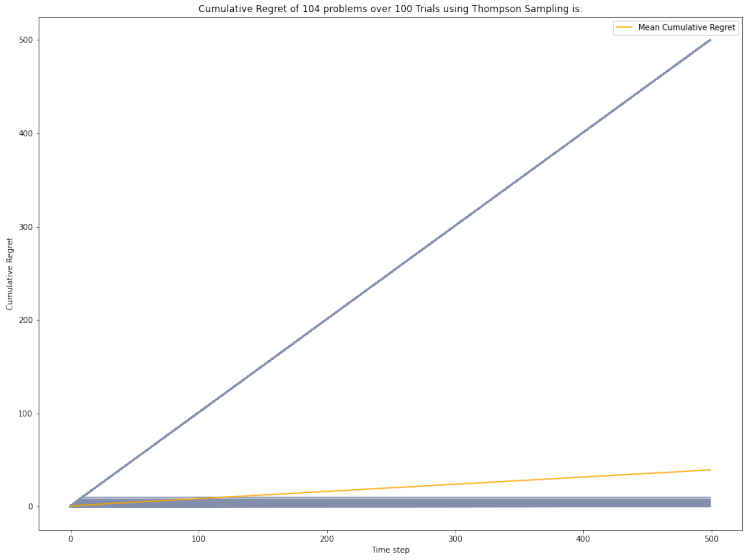
applicable to actual production use. In summary, even though UCB2 is a good choice for low conversion rate scenarios, Thompson Sampling is more of an applicable choice that not only in scenarios with higher baseline conversion rate or higher expected effect size but also in the actual production usage.

4.1.10 Discussion

As demonstrated in the experiments, different bandit algorithms have different performances on the dataset. All algorithms will be able to show the ability to find the best tutor strategy over time. Thompson Sampling, or UCB2 or standard softmax have the best result in terms of minimizing the regret.

With this, it is worth trying to use bandit algorithms for tutor strategy recommendation. This would be an answer to RQ4. However, this experiment only compares the performance among bandit algorithms and these bandit algorithms are

Figure 4.7: Cumulative regret of Thompson Sampling over 100 trials



context-free. The next chapter will explore if the bandit algorithms will introduce bias compare to using random trial methods.

Figure 4.8: Cumulative regret comparison of all bandits

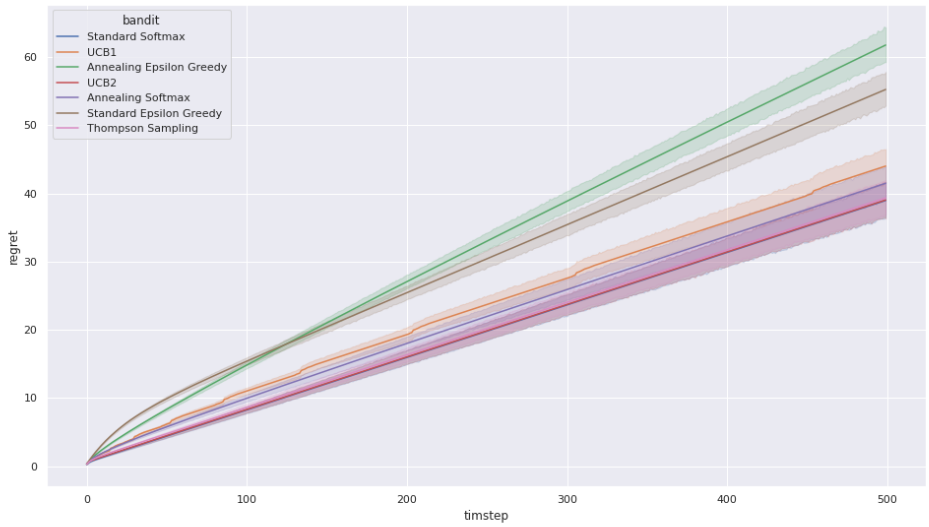
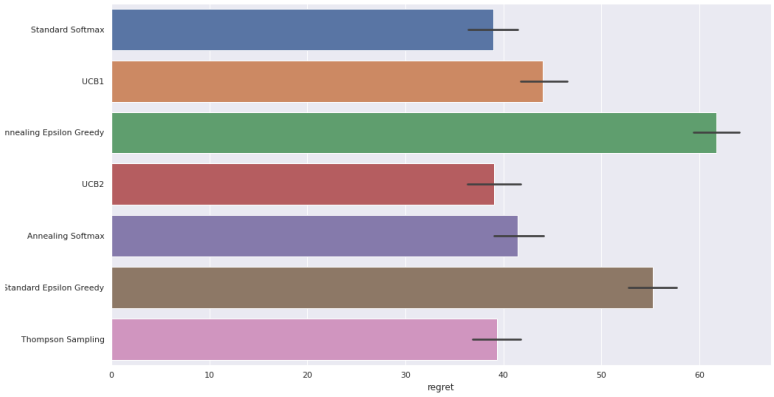


Figure 4.9: Final mean cumulative regret comparison of all bandits



Chapter 5

Experiments

5.1 Comparing Bias and Regret in Reinforcement Learning and Randomized Controlled Trials

The learning science community is perpetually developing new instructional interventions aimed at increasing student engagement and learning. Typically, these new interventions are evaluated using randomized controlled experiments in which students are placed into a control condition, where they receive no intervention, or one of potentially many treatment conditions, where they receive an intervention. These experiments can have a set end date, or they can run until a statistically significant result is observed. An alternative to this paradigm is to use reinforcement learning to provide interventions to students instead of random selection, and to reward the reinforcement learning algorithm when students show positive signs of learning and engagement. While this alternative is optimized for providing students with the interventions most likely to help them succeed as quickly as possible, the lack of randomized analysis creates the opportunity for false assertions and bias in the reinforcement learning algorithms assignment of treatments.

5.1.1 Dataset

The data this experiment use is the same from the last experiment on testing different multi-armed bandit algorithms.

5.1.2 Preprocessing

Apart from the data processed in 4.1, the student features data is also added as in Table5.1.

student_id	s_answer_count	s_mean_score	s_mean_tot	s_mean_attempts	s_mean_frt
14	53	0.408163	32.467074	1.037736	27.698113
18317	9	0.333333	25.602220	1.000000	14.666667
21421	31	0.741935	13.835258	1.161290	8.903226
21432	1	1.000000	24.478766	1.000000	25.000000
...	...	...	...	...	...
21825	67	0.656716	22.876940	1.194030	16.731343
22673	12	0.750000	27.291382	1.083333	23.500000
23098	7	0.714286	21.207534	1.000000	21.142857
24278	56	0.711538	35.037859	1.017857	24.571429

Table 5.1: Processed additional student feature data samples

After concatenating the student features with the processed data, the unique problem number is 2564 and the total samples is 126744. The problem with at least 2 tutor strategies options is 327. And the tutor strategies number is 685 with a unique student number of 9046.

5.1.3 Methodology

Randomly assign tutor strategies randomly to students on a problem level showed great promise. By requesting on-demand assistance, students are more likely to correctly answer the next problem on the first attempt attempt[PH20].

Inspired by the experiment, the way to find out if the MABs will introduce bias in recommending tutor strategies is to conduct random controlled experiment that compared different tutoring strategies available to simulate the effects of us-

ing MABs. Comparing these options revealed that the MAB was able to increase student's correctness on future problems but also led to analysis that found more reliable differences between the available tutoring than the randomized controlled experiment found.

For the random controlled experiments, the two designed methods are:

1. Pure random

   The method is intend to assign randomly tutor strategies all the way.

2. Random until p significant

   The method is intend to assign randomly tutor strategies until there appears to be a significantly better tutor strategy among all available ones. In order to achieve this, after each time that a tutor strategy is given and the reward is observed, performing a t-test on the reward data and calculate the p-value. Once the p-value is less than 0.05, the tutor strategy that has the best effect so far is considered significantly better than the rest and for the remaining time steps, the best one will be the only recommendation.

To compare with the MABs, UCB1 and LinUCB are selected for that they are the counterparts of non-contextual and contextual MAB.

Specifically, the experiment set up is similar to the reinforcement learning setup used in Chapter 4's experiments. For each of the 327 problems and a time step of 500, the experiment will collect the cumulative reward of using pure random method, random until p significant method, context-free bandit algorithm and contextual bandit algorithm. Also the experiment will be run 5 times in order to reduce the random effect.

After each experiment, the accumulative reward will be calculated as well as the best tutor strategy selected by each of the 4 methods. By comparing the reward and fraction is agreement of the best tutor strategies for each problem, the answer to RQ5 will be obtained.

5.1.4 Results

Table 5.2 and 5.3 demonstrates the result of the experiments, note that the NaNs means that there is no statistical best tutor strategies of the given trials using the specific method.

trial_num	problem_id	average_correctness	available_ts	random_until_p_best	random_until_p_reward
0	1057696	0.498	2	NaN	249.0
0	1057697	0.624	2	NaN	312.0
0	1057698	0.924	2	1178526.0	462.0
0	1198578	0.840	2	NaN	420.0
0	1214706	0.552	2	NaN	276.0
...	...	...	...	...	...

Table 5.2: Result of problem-level best tutor strategy data samples

Here, trial_num is the trial id (0-4), problem_id is the problem id,average_correctness is average correctness among all the collected samples, available_ts is the number of available tutor strategies of the problem, random_until_p_best denotes the best tutor strategy of using random until p significant method random_until_p_reward is the accumulative reward of random until p significant method.

linucb_best	linucb_reward	ucb1_arm	ucb1_reward	pure_random_best	pure_random_reward
1257187.0	252	NaN	228	NaN	234
NaN	291	1178525.0	295	NaN	275
NaN	452	NaN	454	NaN	457
NaN	452	NaN	454	NaN	457
NaN	411	NaN	413	NaN	408
NaN	273	NaN	284	NaN	284
...	...	...	...	...	...

Table 5.3: Result of problem-level best tutor strategy of using different methods

Here, linucb_best denotes the best tutor strategy using contextual bandit, linucb_reward is the accumulative reward using contextual bandit, ucb1_arm denotes the best tutor strategy using noncontextual bandit, ucb1_reward accumulative reward using noncontextual bandit , pure_random_best denotes the best tutor strategy using pure random method, pure_random_reward is accumulative reward using pure random method.

By listing the best tutor strategy of different method, the RQ1 can be answered.

The results is showed in Table 5.4. For the selected problems different methods provides generally the same result. For this particular setup, the answer for RQ5 is that does not introduce bias.

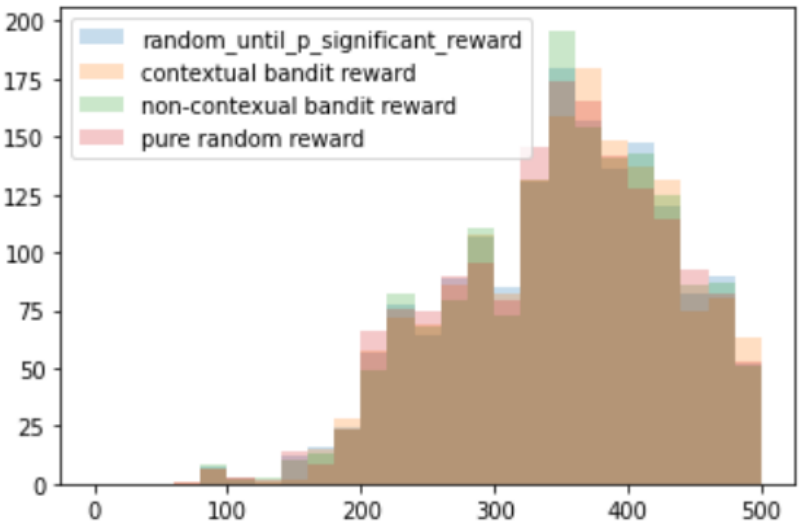
Method name	Fraction in agreement:
random_until_p_significant	0.6886850152905198
contextual_bandit	0.7363914373088685
noncontextual_bandit	0.728440366972477

Table 5.4: Fraction in agreement of best tutor strategy

Also by analysing the rewards in Figure5.1, the 4 different methods does not appear to have very significant differences which means all other methods are not showing significant advantage over just randomly assign the different tutor strategies every time.

Although the result is not as idea as the assumption that MABs will show significant advantage, the experiments do demonstrate that MABs have the ability identify the best tutor strategies. The limitation might come from the fact that most of the problems only have two available tutor strategies and the sample number of the given the different tutor strategy is almost equal. Another possible factor is that the tutor strategies are created by trusted teachers, the quality of the tutor strategies is almost the same for helping the student answer the next problem correctly.

Figure 5.1: Reward comparison between differnt methods



With this assumptions in mind, the next step would be introduce more tutor strategies for the problems and gather more data. Once the data is large enough, the MABs might have significant advantage over just randomly assign tutor strategies.

Chapter 6

Summary

This thesis explores the possibility and effectiveness of utilizing Multi-armed bandits for recommending different tutor strategy. This thesis intruded the design and implementation of the infrastructure that can support to provide tutor-strategy recommending service for ASSISTments. The infrastructure is designed to integrated seamlessly into ASSISTments ecosystem.

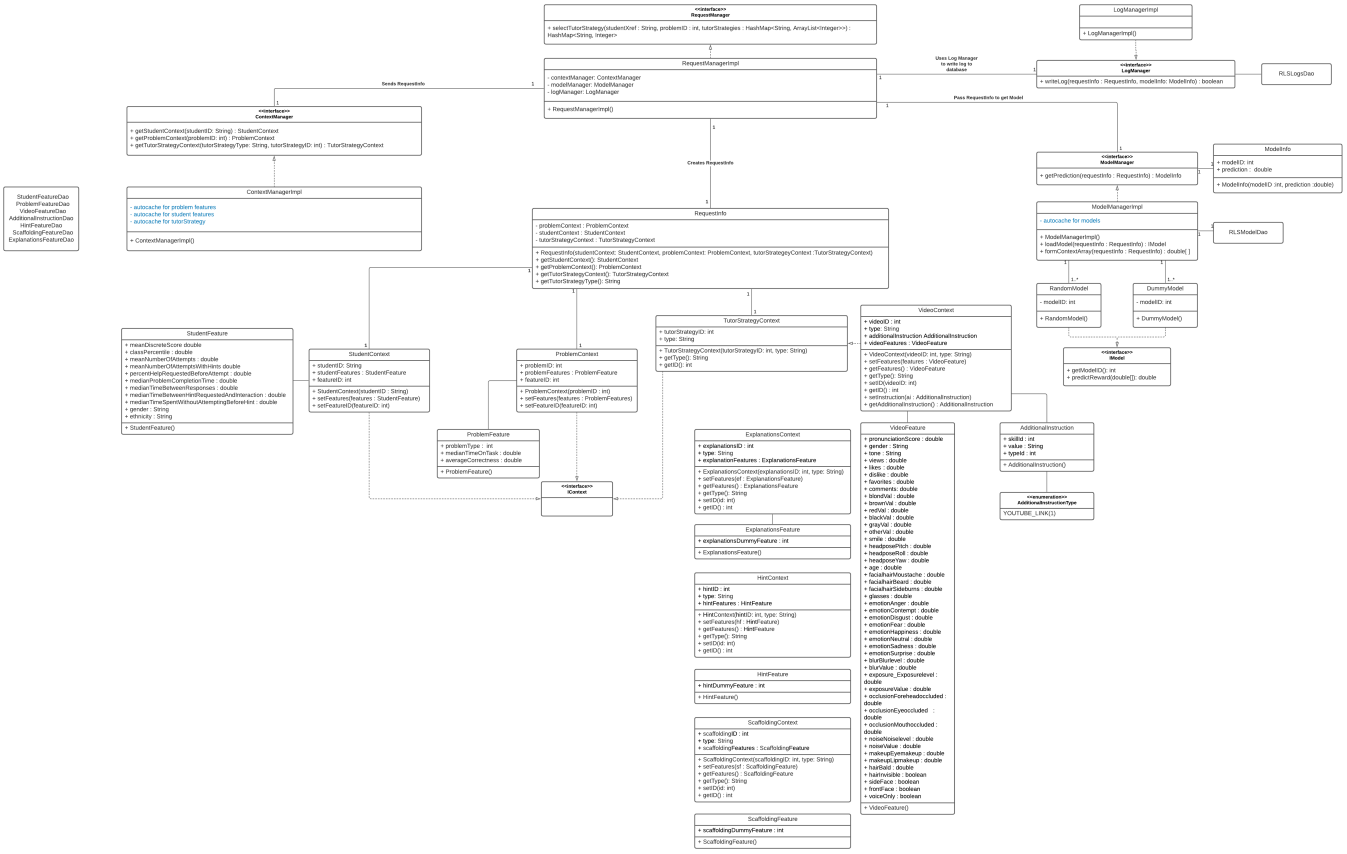
This thesis achieved the goals by answering proposed research questions. For RQ1, as showed in experiments from Chapter 4 and 5, the MABs were able to discover the best tutor strategy for the specific problem. For RQ2, by introducing student features created, the contextual bandit algorithm is also able to provide the best recommendation for the students. For RQ3, in feature creation, the important features were selected on both student-level and problem-level. For RQ4, comparing different bandit algorithms using the mean cumulative regret as the metric. And the results suggests that Thompson Sampling is the best choice for actual production usage. For RQ5, the comparison between different random control experiments with MABs on the specific setup and found that the bandit algorithms does not introduce bias.

Appendix A

Class Diagram

A.1 Class diagram

Figure A.1: RLS class diagram



Bibliography

[ACBF02] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[AG13] Shipra Agrawal and Navin Goyal. Thompson sampling for contextual bandits with linear payoffs. In *International Conference on Machine Learning*, pages 127–135, 2013.

[BFT20] Erik Black, Richard Ferdig, and Lindsay A Thompson. K-12 virtual schooling, covid-19, and student success. *JAMA pediatrics*, 2020.

[BLDC16] Sameer Bhatnagar, Nathaniel Lasry, Michel Desmarais, and Elizabeth Charles. Dalite: Asynchronous peer instruction for moocs. In *European Conference on Technology Enhanced Learning*, pages 505–508. Springer, 2016.

[But18] Andrew C Butler. Multiple-choice testing in education: Are the best practices for assessment also good for learning? *Journal of Applied Research in Memory and Cognition*, 7(3):323–331, 2018.

[DLRH08] Paul Denny, Andrew Luxton-Reilly, and John Hamer. The peerwise system of student contributed assessment questions. In *Proceedings of the tenth conference on Australasian computing education-Volume 78*, pages 69–74. Citeseer, 2008.

[FSA+18] Fareeha Farooqui, Nadia Saeed, Sahira Aaraj, Muneeza A Sami, and Muhammad Amir. A comparison between written assessment methods: Multiple-choice and short answer questions in end-of-clerkship examinations for final year medical students. *Cureus*, 10(12), 2018.

[HH14] Neil T Heffernan and Cristina Lindquist Heffernan. The assistments ecosystem: Building a platform that brings scientists and teachers together for minimally invasive research on human learning and teaching. *International Journal of Artificial Intelligence in Education*, 24(4):470–497, 2014.

[Hoe94] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

[HSH+12] Gwo-Jen Hwang, Han-Yu Sung, Chun-Ming Hung, Iwen Huang, and Chin-Chung Tsai. Development of a personalized educational computer game based on students' learning styles. *Educational Technology Research and Development*, 60(4):623–638, 2012.

[JSB18] Yuchao Jiang, Daniel Schlagwein, and Boualem Benatallah. A review on crowdsourcing for education: State of the art of literature and practice. In *PACIS*, page 180, 2018.

[KLM96] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[KVJ87] Michael N Katehakis and Arthur F Veinott Jr. The multi-armed bandit problem: decomposition and computation. *Mathematics of Operations Research*, 12(2):262–268, 1987.

[LCLS10] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670, 2010.

[LS82] Graham Loomes and Robert Sugden. Regret theory: An alternative theory of rational choice under uncertainty. *The economic journal*, 92(368):805–824, 1982.

[PH20] Thanaporn Patikorn and Neil T Heffernan. Effectiveness of crowdsourcing on-demand assistance from teachers in online learning platforms. In *Proceedings of the Seventh ACM Conference on Learning@ Scale*, pages 115–124, 2020.

[PSBH15] John F Pane, Elizabeth D Steiner, Matthew D Baird, and Laura S Hamilton. Continued progress: Promising evidence on personalized learning. *Rand Corporation*, 2015.

[RVRK+17] Daniel Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on thompson sampling. *arXiv preprint arXiv:1707.02038*, 2017.

[RYW19] Anna Rafferty, Huiji Ying, and Joseph Williams. Statistical consequences of using multi-armed bandits to conduct adaptive educational experiments. *JEDM— Journal of Educational Data Mining*, 11(1):47–79, 2019.

[SDW+18] Avi Segal, Yossi Ben David, Joseph Jay Williams, Kobi Gal, and Yaar Shalom. Combining difficulty ranking with multi-armed bandits to sequence educational content. In *International conference on artificial intelligence in education*, pages 317–321. Springer, 2018.

[TL00] Sebastian Thrun and Michael L Littman. Reinforcement learning: an introduction. *AI Magazine*, 21(1):103–103, 2000.

[WAC+12] Daniel S Weld, Eytan Adar, Lydia B Chilton, Raphael Hoffmann, Eric Horvitz, Mitchell Koch, James A Landay, Christopher H Lin, and Mausam Mausam. Personalized online education-a crowdsourcing challenge. In *HCOMP@ AAAI*. Citeseer, 2012.

[WS17] Jacob Whitehill and Margo Seltzer. A crowdsourcing approach to collecting tutorial videos–toward personalized learning-at-scale. In *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale*, pages 157–160, 2017.