Study the use of AI and machine learning to model and support learners and researchers

Liu Rongxing

4th Year Project Report
Computer Science and Mathematics
School of Informatics
University of Edinburgh

2022

Abstract

The rapid development in E-learning has provided students with more flexible learning tools. These AI-powered platforms allow for a more personalized learning experience that adapts to the user's changing pedagogical demands over a course. In this report, we introduce two ways to build tools for modelling students when solving various questions online. One is an algorithm called EduRank that serves as a recommender system by constructing a difficulty ranking for each student. It was tested on two datasets. The other is a Reinforcement Learning (RL) based approach that helps sequence learning materials to students by maximizing expected future performance. EduRank was trained and tested on two datasets: PSLC and Ednet; and it obtained an average precision(AP) score of 0.57 and 0.54 respectively. The RL tasks used two approaches: the model-based and model-free methods. For the model-based methods, seven most significant features were extracted to represents states in the MDPs. For the model-free methods, nine experiments of CQL and corresponding FQE were run. In addition, perturbations to strong, intermediate and weak students were performed to test models' robustness. We also analyzed the details of each policy by calculating the number of questions in various difficulty levels given to different student groups. The question allocation is reasonable as strong students are given with more difficult questions while weak students are given more lectures to watch.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Liu Rongxing*)

Acknowledgements

The task of reinforcement learning is difficult for me at the beginning since I do not have any prior knowledge about it. However, Aqil's work is very detailed and understandable. It is a very inspiring work indeed and I wish I can continue working in the related fields in the future. In addition, I must appreciate the support from the informatics department. The offline training would never be possible without access to the teaching clusters. Last but not least, I also have to thank my supervisor Kobi Gal and Avi Segal that help me overcome all the obstacles.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

Teaching and learning have been changed as a result of technological advancements. Besides notes and lectures provided by schools, student can also choose a much wider variety of learning materials online. On the other hand, choosing the most suitable learning methods has been a new challenge for students and teachers. Some students may spend unproportionate amount of time online yet still struggling to find useful and helpful materials; some may spend a lot of time on practices yet not getting desirable grades for exams. This has driven the demand for recommender systems and intelligent tutoring systems. The recommender system will rank a number of questions for a student and offer the questions in increasing level of difficulties so that the student can improve learning progressively, neither discouraged by hard questions nor bored by easy questions. The intelligent tutoring system will sequence learning materials (such as videos and questions) to students with the objective of maximizing learning gains. This ensures that students can use their time more efficiently.

1.2 Objectives

In this project, we are exploring a difficulty ranking algorithm called EduRank and an adaptive RL based pedagogical agent. EduRank predicts a personalized difficulty ranking for each student by aggregating the rankings of similar students using different aspects of their performance on common questions. Rather than sorting the questions according to the student's predicted score, it infers a difficulty ranking directly over the questions for each student. Its performance was evaluated on two datasets. The first one was published in the KDD cup 2010 by the Pittsburgh Science of Learning Center (PSLC). It contains around 800,000 answer attempts by 575 students and was collected during 2005-2006. The second one, Ednet, is a large collection of student activities from Santa, an online learning platform with intelligent tutoring system. The metric for evaluation is Average Precision (AP) score. The RL based approach is also experimenting on Ednet. It derives several learning policies and test their robustness under the perturbation of different features. We split the students into three groups:

strong, intermediate and weak. The effect of perturbation was tested on all three groups. The policies are evaluated by Expected Cumulative Reward (ECR), which is the average expected return from the initial state.

1.3 Document Structure

The structure of this report is as follows: the next chapter is literature reviews of several previous works on applications of machine learning in education; Chapter 3 is about the introduction, mechanism and implementation of EduRank; Chapter 4 is an outline of previous work by Aqil that includes the RL agent employed, with results and analysis for various experiments; Chapter 5 is about the extensions we have done further to Aqil's work; Chapter 6 is the conclusion to the studies and Chapter 7 is the Appendix that contains some additional useful information.

Chapter 2

Literature Review

There have been many related work in this area. In 2014, Avi Segal et al. [13]introduced EduRank that combines collaborative filtering algorithms with social choice theory to personalize educational content for students. The algorithm was evaluated using two ranking scoring metrics— NDPM and AP. Its performance was compared with other known methods such as EigenRank Algorithm, User Based Collaborative Filtering (UBCF), a matrix factorization method using SVD, an expert ranking (denoted by CER) and Topic-Based Ranker (TBR) [3]. EduRank achieved the best result on both datasets and both metrics. However, the execution of this algorithm is costly due to its high-order complexity nature.

In 2019, Avi Segal et al[12]. improved upon his previous work on EduRank [4]. The algorithm was extended to handle the "cold-start problem" by incorporating a prior score $pr(q_k)$ for every question q_k in the training set by averaging over the scores of all students that solved this question in the training set. The results showed that the EduRank+Prior method beat the traditional EduRank algorithm, with noticeable differences up to the eighth week of training. Furthermore, the algorithm was also compared with ASC sequencing approach based on pedagogical experts on their performances in classrooms. The results showed that students using EduRank approach solved more difficult questions and achieved higher grades, and having to solve more difficult questions did not discourage students' motivation to use the e-learning system. In addition, the research showed there is little agreement among students in difficulty rankings for the same set of questions, emphasizing the need for personalization in sequencing questions. However, the experiment overlooked the possibility that students can seek for assistance on a difficult question which makes their scores higher, and time spent lower. Solving a question with assistance does not improve as much as solving a question individually.

In 2016, Yossi Ben David et al[5]. introduced a Bayesian Knowledge Tracing model for sequencing questions to students. It is based on using knowledge tracing to model students' skill growth over time and to choose questions that will improve their learning within the model's range of abilities. The model was shown to outperform other models that did not take into account partial credit scores, penalty on multiple attempts or integrating item difficulty. They also incorporated the model into a sequencing

algorithm and deployed in classrooms. It outperformed the ASC sequencing approach as shown by students' scores and engagement. However, the research did not include pre and post tests to measure the effects of the algorithms on students' performance and learning gains more accurately.

In 2018, Kobi Gal et al.[11] introduced a new computational approach to this problem called MAPLE (Multi-Armed Bandits based Personalization for Learning Environments) that combines difficulty ranking with multi-armed bandits [7]. MAPLE was compared with three other sequencing algorithms, namely the Ascending approach that sequenced questions according to an absolute difficulty ranking that was determined by pedagogical experts, the EduRank approach, and the Naive Maple approach that sequenced questions using the multi-armed bandit algorithm with random weights initialization (without the EduRank based difficulty ranking component). The results showed that MAPLE adapted well to students in various skill competency levels (the weak students with level under 0.33, the average students with level between 0.33 and 0.67, and the strong students with level above 0.67), giving more questions close to their abilities. In addition, MAPLE gave the largest increase in median of student competency level for both average students and strong students. However, the models did not do well on weak students. This could be due to an overestimates of weak students' learning rate.

In 2021, Aqil[1] introduced a Reinforcement Learning agent for optimizing sequencing of learning materials. The training was conducted offline using the EdNet dataset. They used a greedy iterative feature augmentation pipeline, with the Expected Cumulative Reward (ECR) metric for assessing policies, to determine the optimal feature representation. In the end, 4 extra features are taken (**expl_received**, **ssl**, **prev_correct**, **av_fam**) besides the three basic features **'correct_so_far','av_time'** and **'topic_fam'**. The research also explored the methods' robustness to perturbations in the environment induced by strong and weak learners and found that larger representations are more robust towards deviations from the expected dynamics derived from the data. Moreover, weak students are more susceptible to perturbations than strong students. One drawback of this RL agent is that it is extremely time consuming, and

Chapter 3

EduRank

3.1 AP Metric

Instead of using accuracy for comparing similarities between two results, we use AP score [19] that would assign more weights to errors in higher-ranked questions. Consider the example shown in Table 3.1: Both proposed rankings have correctly predicted the position of three questions. The first one correctly predicted $q2, q3$ and $q4$ while the second one correctly predicted $q1, q2$ and $q3$. Hence, they have the same accuracy. However, if we look at the mistakes, the first ranking predicts the hardest question $q1$ as the easiest. As a result, $q1$ will be recommended to the student at a very early stage and this will strongly discourage student's motivation. In contrast, the mistake of the second ranking only appears at the easiest two questions which in reality makes little difference to the student. Hence, in this case we would prefer the second ranking.

Table 3.1: An example to illustrate AP score, questions ranked in decreasing difficulty level

Reference Ranking	Proposed Ranking 1	Proposed Ranking 2
q1	q5	q1
q2	q2	q2
q3	q3	q3
q4	q4	q5
q5	q1	q4

With this in mind, we now start our formal definition of AP score. Let $\prec$ be the order over a question set Q. For $q_k, q_l \in Q, q_k \prec q_l \iff q_k$ is more difficult than q_l. Let $\prec_i$ be a restriction over $L_i \in Q$ such that $q_k \prec_i q_j \iff q_k \prec q_j$ and $q_k, q_j \in L_i$. Let $\prec_1$ be the reference ranking and $\prec_2$ be the proposed ranking. We say that $\prec_1$ and $\prec_2$ agree on $q_j, q_k \iff q_j \prec_1 q_k$ and $q_j \prec_2 q_k$. For a given question $q_k \in L$, define

$$Z^k(L, \prec_2) = \{(q_j, q_k) | k \neq j, q_k \prec_2 q_j \text{ and } q_k, q_j \in L_i\} \tag{1}$$

These are all the questions harder than q_k. Let $I(q_j, q_k, \prec_1, \prec_2)$ be an indicator function such that

$$I(q_j, q_k, \prec_1, \prec_2) = \begin{cases} 1 & \prec_1 \text{ and } \prec_2 \text{ agree on } q_j, q_k \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

We define the normalized agreement score $A_k(L, \prec_1, \prec_2)$ as

$$A_k(L, \prec_1, \prec_2) = \frac{1}{k-1} \sum_{(q_j, q_k) \in Z^k(L, \prec_2)} I(q_j, q_k, \prec_1, \prec_2) \tag{3}$$

Then the AP score $S_{AP}(L, \prec_1, \prec_2)$ can be calculated by taking the average of A_k for all $k > 1$

$$S_{AP}(L, \prec_1, \prec_2) = \frac{1}{|L| - 1} \sum_{k=2}^{|L|} A_k(L, \prec_1, \prec_2) \tag{4}$$

Intuitively, for larger values of k, the summation of all agreements preceding q_k will be divided by a larger value, making them less significant while for smaller values of k, the denominator is also smaller and the characteristics of those agreements will be mostly preserved.

3.2 Algorithm

We have now formalized the definition of AP score. Next, we will show how to use it in the EduRank Algorithm.

The difficulty ranking problem is defined as follows: given a question set Q, a student set S, each student $s_i \in S$ with a subset T_i of Q (which are questions attempted by s_i, in the real context they could be questions in a tutorial sheet), and a known ranking $\prec_i$ over T_i, we need to find a difficulty ranking $\hat{\prec}$ over $L \in Q$(which are questions he has yet to solve, in the real context they could be questions in an assignment) for a target student s. To do this, we are comparing the target student's ranking with other students' ranking, calculating their similarities and use the known ranking of similar students to predict a ranking for the target student.

Firstly, we define the win score $\gamma(q_k, q_l, \prec)$ over q_k, q_l given ranking $\prec$ as

$$\gamma(q_k, q_l, \prec) = \begin{cases} 1 & q_k \prec q_l \\ -1 & q_l \prec q_k \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The relative voting $rv(q_k, q_l, S)$ of q_k, q_l for a target student i given a group of students S and their difficulty rankings is

$$rv(q_k, q_l, S) = sign(\sum_{j \in S \setminus i} S_{AP}(T_i, \prec_i, \prec_j) \dot{\gamma}(q_k, q_l, \prec_j)) \tag{5}$$

Next, for each $q \in L_i$ we can calculate its Copeland score $c(q, S, L_i)$ by summing over relative voting between q and all the other questions $q_l \in L_i \setminus q$

$$c(q, S, L_i) = \sum_{q_l \in L_i \setminus q} rv(q_k, q_l, S) \tag{6}$$

Finally, we will rank questions in L_i by decreasing Copeland score, i.e. questions with higher Copeland score is considered more difficult. The details of the algorithm are shown below.

Algorithm 1 EduRank(S,Q,$\prec$,i,L_i)

Input: Set of students S; Set of questions Q; For each student $s_j \in S$, a partial ranking $\prec_j$ over $T_j \in Q$; Target student $s_i \in S$; Set of questions L_i to rank for s_i

Output: a partial order $\hat{\prec}_j$ over L_i

1: **for** $q \in L_i$ **do**
2: $\quad c(q,S,L_i) = \sum_{q_l \in L_i \setminus q} rv(q_k, q_l, S)$
3: **end for**
4: $\hat{\prec}_j = \{\forall q_k, q_l \in L_i, q_k \hat{\prec}_j q_l \iff c(q_k) > c(q_l)\}$
5: **Return** $\hat{\prec}_j$

The time complexity of EduRank Algorithm is $O(|S| \cdot max(|T_i|^3, |L_i|^2))$. For the target student s_i, we need to calculate the AP score of s_i with all the other students. In each calculation, we need to find $Z^k(L, \prec_2), A_k(L, \prec_1, \prec_2), S_{AP}(L, \prec_1, \prec_2)$ sequentially and the complexity in each step is $O(T_i)$. Hence, the overall complexity is $O(|S| \cdot |T_i|^3)$. On the other hand, the computation of relative voting involves calculating the win score $\gamma(q_k, q_l, \prec_j)$ for each question pair (q_k, q_l) and each student $j \in S \setminus i$, and the complexity in this case will be $O(|S| \cdot |L_i|^2)$. Since the AP score and relative voting can be calculated separately, we shall only take into account the maximum of $O(|T_i|^3)$ and $O(|L_i|^2)$ for deciding the complexity. This means that if we need to find a difficulty ranking for a much larger question set, $O(|L_i|^2)$ will supercede $O(|T_i|^3)$. However, as we will discuss later, the complexity of EduRank in our case can be considered to be $O(|S| \cdot |T_i)^3$.

3.3 Implementation

EduRank was implemented and tested on two datasets, PSLC[7] and Ednet[2]. For the PSLC dataset, we removed students that solved less than 100 questions to ensure that there is enough information for the algorithm to work well. This left us 172 students. Then for each student s_i, we inferred a reference ranking $\prec_i$ for all questions Q_i the student attempted. We begin by ranking questions based on the number of incorrect attempts. A larger number incorrect attempts suggests that the question is more difficult to the student. Then we break ties by the total time taken.

After obtaining the reference rankings, we split the questions into two set of equal size, one for training the other for testing. The ranking of questions in training set is given as input and we used EduRank to predict a difficulty ranking on questions in test set. The predicted ranking is then compared with the reference ranking and their AP score is computed.

Unfortunately, due to the giant size of Ednet KT1 dataset that consists of data from over 780,000 students and the bottleneck complexity of EduRank, we cannot afford to run the algorithm on all students. To make computation realistic, we selected students who solved exactly 100 questions then we randomly chose twenty percent of those students.

Even by doing this, we are still left with 239 students, which is more than that from PSLC. Next we used the same method mentioned above to infer reference ranking, split questions and compare results.

3.4 Results

It took about one and a half hours to run EduRank for all students in the PSLC dataset and about ten minutes in the case of Ednet KT1. Both were done on a DICE computer. We took the average of AP score over all students for both datasets. The average AP score is 0.57 for PSLC and 0.54 for Ednet KT1. The reason for the worse performance on Ednet KT1 could be subject to bias sampling. It is difficult to find a suitable sample that is both representative and cost-efficient in computation. In addition, there are other features in higher hierarchy of Ednet that KT1 does not contain. For example, KT3 also contains information about various learning activities, such as reading through teachers' commentary on a question or watching lectures on a specific topic. Engaging in such activities will naturally increase students' accuracy in answering questions. However, that does not fully reflect students' true capabilities. It is likely that a student is unfamiliar with a topic, thus he starts to watch lectures and after that get descent results, while after some time his grasp fades and the topic may become obscure to him again. This is one of the limitations of EduRank which the RL method introduced later in this report will address.

Chapter 4

RL Based Approach

4.1 RL Background

Reinforcement Learning is another aspect of machine learning alongside with supervised, semi-supervised and unsupervised learning[16]. It uses an exploration-exploitation strategy to find actions that maximize the reward. The agent has to exploit actions that are known to produce high reward, and explore new actions that may potentially increase the reward. It interacts with the environment by performing an 'action' from a finite range of options. This yields a reward value and a transition to the next state where the process will repeat.

Our RL agent is based on a finite Markov Decision Process(MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S}$ is the set of all possible states, $\mathcal{A}$ is the set of all actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ is all transition probabilities from state $s_i \in \mathcal{S}$ to state $s_j \in \mathcal{S}$ via action $a_k \in \mathcal{A}$, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ contains the reward functions of such transitions. An MDP is a RL task that satisfy the markov property, meaning that what will happen in the future depends only on the current state and not any states before. The outline of MDP is shown in 4.1. At time t, the agent constructs a mapping π_t from S_t to the probabilities of selecting each possible A_t, which is denoted by $\pi_t(A_t = a|S_t = s)$. This is called the agent's policy. The policy changes as a result of the agent's experience, as defined by RL methods. There is an associated reward R_{t+1} after each action, which is

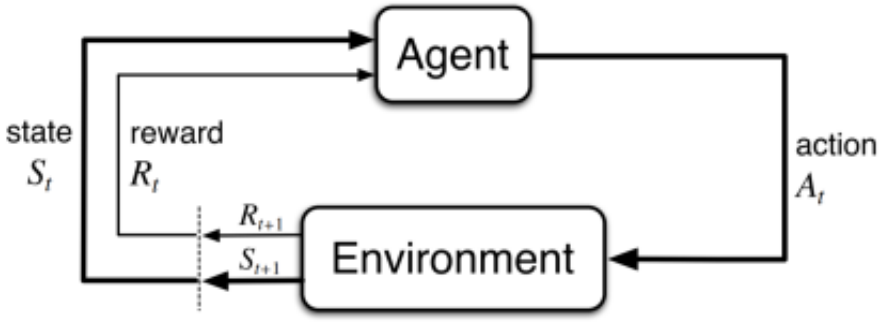

Figure 4.1: Illustration of MDP framework

normally multiplied by a discount factor $\gamma \in [0,1]$ raised to the power of t. The size of γ determines how much emphasis the agent puts on rewards in the far future relative to those in the near future. As t increases, γ^t decreases, making the reward less significant. $\gamma = 0$ means that the agent is completely ignorant about future reward while $\gamma = 1$ means that the agent treats each time step equally important. The agent's purpose is to maximise the discounted cumulative reward (also known as return) it receives over time. The return of a state under a policy π is calculated by equation 4.1.

$$\mathbb{E}_{a\sim\pi,r,s\sim\mathcal{P}}[\sum_{t=0}^{\infty}\gamma^t \cdot r(s_t,a_t)|s_0 = s] \tag{4.1}$$

4.2 Data Description

The data we used is Ednet[2]. It is organized into a hierarchical structure in which each hierarchy contains different level of information. Specifically, it was divided into four datasets named KT1, KT2, KT3 and KT4. KT1 consists of the rudimentary information of students' learning session, namely the question-response pairs and elapsed time. Various deep-learning knowledge tracing models such as s Deep Knowledge Tracing [16] use this format. KT2 makes an improvement over KT1 in the sense that it not only records students' final answers but also captures candidate answers that students hesitate on. KT3 incorporates information on students' learning activities other than solving questions. Such information can be used to explore effect on students' learning state. For instance, one can analyze the time each student spends watching lectures and observe how it impacts different learning behaviours and performance. KT4 contains the most detailed information including playing and pausing lectures. An example of KT1 dataset is shown in 4.2[10]. Examples of the other three datasets can be found in Appendix 5.2-5.4.

timestamp	question_id	bundle_id	user_answer	elapsed_time
1548996377530	48	q2844	d	47000
1548996378149	48	q2845	d	47000
1548996378665	48	q2846	d	47000
1548996671661	49	q4353	c	67000
1548996787866	50	q3944	a	54000

Figure 4.2: Example student data in EdNet-KT1.

Besides the hierarchical data, Ednet also contains a total 13,169 questions and 1,021 lectures tagged with 293 skill types. Examples are shown in Appendix 5.5-5.6[10]. Each lecture has one corresponding tag annotated by experts while each question may have more tags. Both questions and lectures are categorized into 7 parts. The distribution of number of questions per part and per skill tag are shown in 4.3 and 4.4.

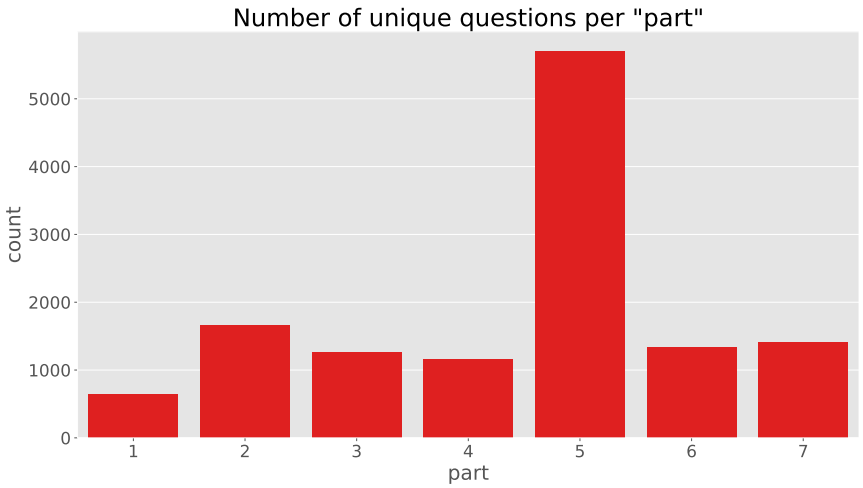

Figure 4.3: Number of unique questions per part.

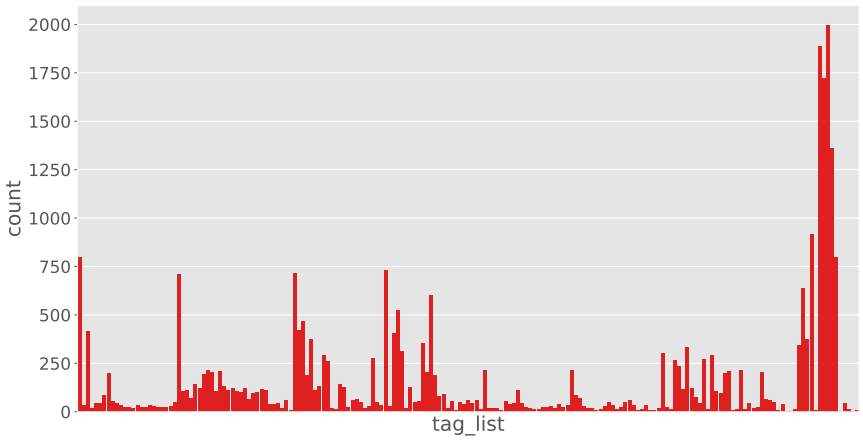

Figure 4.4: Number of unique questions per skill tag.

4.3 Feature Extraction

With the background knowledge and an overview of the data in mind, we now need to find the state set $\mathcal{S}$, action set $\mathcal{A}$, and reward function $\mathcal{R}$ which will be used to define the MDPs.

4.3.1 States

We established a feature pool from which several representations are created using the greedy iterative augmentation approach to evaluate the influence of the representation on performance. We started with three basic features derived from KT1, namely **'correct_so_far','av_time'** and **'topic_fam'**. **'topic_fam'** represents the familiarity of previously chosen part of a topic by students; **'correct_so_far'** is the quantized ratio

of correct responses to total number of responses; and **'av_time'** measures the average time taken by students on each topic. Besides these three features, we also considered 7 other features shown in Table 4.1. These features together address the limitations of EduRank mentioned in 3.4. For example, **'ssl'** can measure user's level of memorization to a topic. **'sec guess'** and **'slow answer'** can reflect user's level of confidence to the question. The number of bins is the number of quantiles the data is splitted into.

Table 4.1: Additional features. Here 'Bins' refers to the number of quantiles

Feature	Bins	Description
prev correct	3	Whether the previous question was answered correctly
expl received	4	Cumulative count of explanations reviewed by the user.
ssl	8	Number of steps since the current part was last encountered
sec guess	4	Number of second guesses to a question
lects consumed	4	Number of lectures watched
slow answer	2	Whether the time taken to answer a question is slower than average
steps in part	4	Number of steps spent

4.3.2 Action

After determining all possible states, our next step is to define the action set. Intuitively, an action in the context of learning must be related to attempting a question or watching a lecture. However, due to the large number of questions and lectures, we cannot make our model so detailed that it will assign a specific question(lecture) to a student. Instead, we will only recommend a topic and a difficulty level of the question(lecture). Students are free to choose any questions within that specific topic and difficulty level.

As shown in 4.3 and 4.4, there is a fairly equal distribution for six parts except part 5. However, the distribution of skill tags is extremely uneven. Hence, 'part' was chosen in order to get equal supporting observations for each class from the dataset. To determine the level of difficulty to a question, we calculated the distribution of percentage correctness for all questions. The result is shown in 4.5. As shown by the blue dotted lines, questions are splitted evenly into four parts by three quartiles. Questions with below 50.6% percentage correctness are 'very hard'; questions with percentage correctness between 50.6% and 61.8% are 'hard; questions with percentage correctness between 61.8% and 72.7% are 'medium'; and questions with percentage correctness above 72.7% are 'easy'.

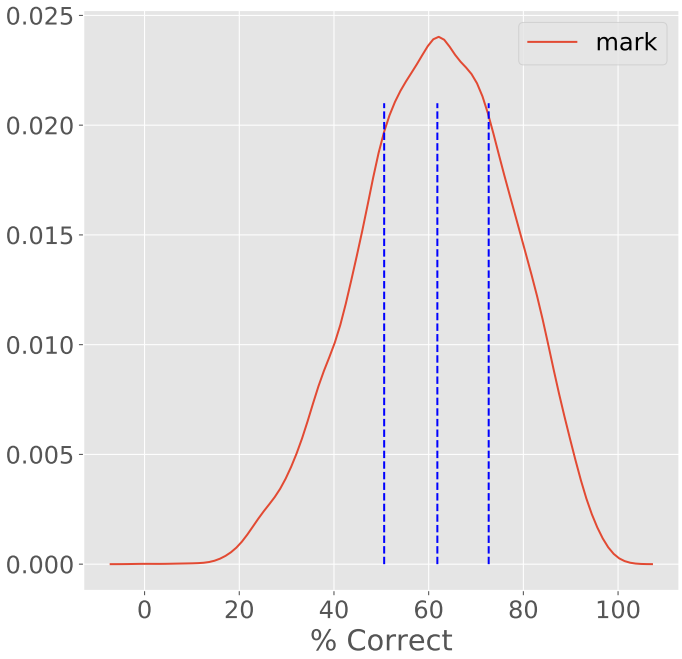

Figure 4.5: Distribution of percentage of questions with respect to their marks.

All lectures are assigned a value of 0 for their difficulty since there is no way to measure a lecture's difficulty. Note that lectures also has two additional parts, with '0' meaning 'general part' and '-1' meaning 'not tagged'. Therefore, we have a total of $5 \times 7 + 2 = 37$ possible actions. The distribution of actions is shown in 4.6

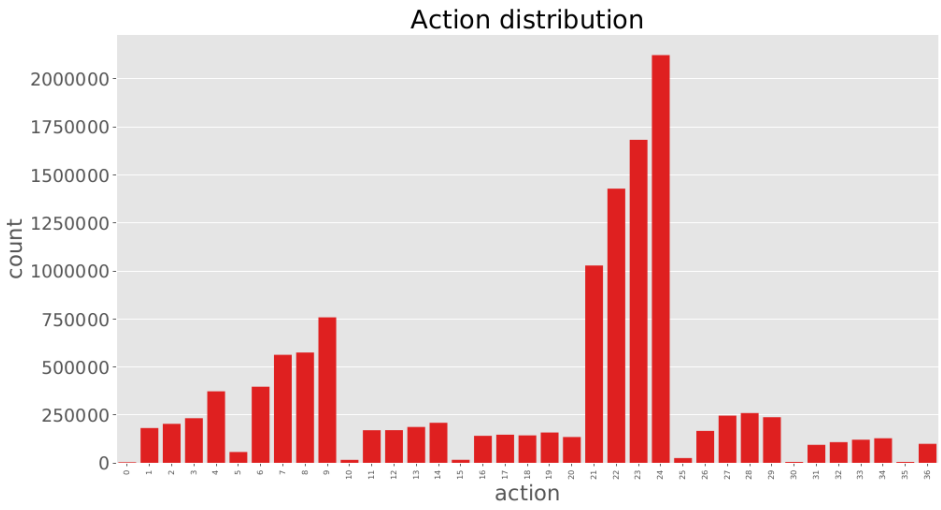

Figure 4.6: Distribution of various actions.

We can see graph the graph that for each part, as the level of difficulty increases, the number of questions students attempted in that level also increases.

4.3.3 Reward

The reward is calculated in such a way that correct responses on harder questions attain higher rewards while incorrect responses on easier questions attain higher penalties. We set four possible rewards [1,4] and four penalties [-4,-1]. Since in general students would get more reward than penalty, the penalty is scaled from [-4,-1] to [-8,-2]. This makes rewards closer to a normal distribution. Other activities such as watching lectures or reading comments have no correct answer, hence the reward is set to 0 throughout.

4.4 Model-Based Offline RL

4.4.1 MDP Derivation

With the feature pool obtained, the next step is to derive an MDP student model. We used maximum likelihood estimate(MLE) to calculate the transition probability

$$\hat{p}(s_j|s_i,a_k) = \frac{c(s_j,s_i,a_k)}{\sum_{j=1}^{n} c(s_j,s_i,a_k)}$$

where $c(s_j,s_i,a_k)$ is the number of occurence of s_j when a_k is applied to s_i. Besides transition probabilities, the other component is the reward function. Instead of using the reward function defined in terms of the three argument dynamics $\mathcal{R}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$[16], in our context we set the reward as a function of the correctness of responses and the level of difficulty to questions. Recall in section 4.2 that the reward can take values $r \in \{-8,-6,-4,-2,0,1,2,3,4\}$. The level of difficulty can be inferred from the (s,a,s') tuple. For example, an increase in the state **correct_so_far** probably implies an easy question while an increase in the state **avg_time** implies a difficult question. The information of correctness can be obtained from the feature **prev correct**. For given s,a, the probability of each possible pair of next state and reward, s',r is defined as[16]

$$p(s',r|s,a) = Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}$$

This can be calculated in the way similar to calculating the transition probabilities. This enables us to compute expected rewards for state-action pairs,

$$r(s,a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s',r|s,a)$$

and the expected rewards for state-action-next-state triples,

$$r(s,a,s') = \mathbb{E}[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s',r|s,a)}{p(s'|s,a)}$$

where $\mathcal{R}$ is the discrete set of all possible rewards $\{-8,-6,-4,-2,0,1,2,3,4\}$.

One drawback of using MLE to estimate transition probabilities is that when the sample size i.e. $\sum_{i=1}^{n} c(s_j,s_i,a_k)$ is small, the result may be unreliable. The state feature representation must be so simple that the small dataset can give adequate support. However, with Ednet being a sufficiently large dataset, we are confident to build a complex enough model. For the base representation with **topic_fam**,**correct_so_far**,**avg_time**, the number of states is shown in 4.7. The minimum number of states is 41,111.

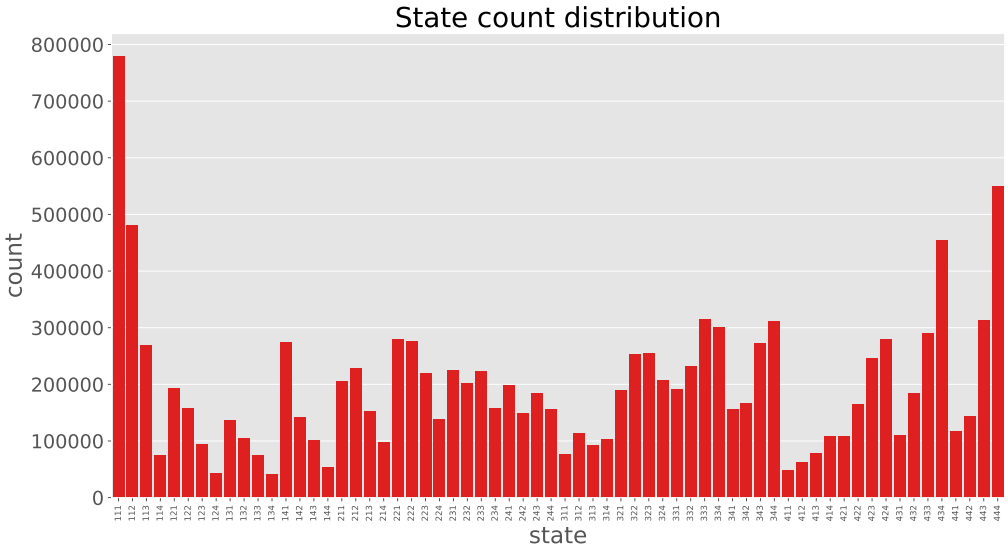

Figure 4.7: Distribution of number of states. There are a total of 64 states since each states are quantized into 4 bins.

4.4.2 Policy Iteration

Next step is to train the agent using the Policy Iteration algorithm[16]. The details are shown in Algorithm 2.

Algorithm 2 Policy Iteration (using iterative policy evaluation)

Input: $\mathcal{S}, \mathcal{A}$,a small value θ

1: Initialise $V(s) \in \mathbb{R}, \pi(s) \in \mathcal{A}(s)$ for all $s \in \mathcal{S}$
2: **Repeat**
3: **for** $s \in \mathcal{S}$ **do**
4: $V(s) = \sum_{s',r} p(s',r|s,\pi(s))[r+\gamma \cdot V(s')]$
5: $\Delta = max(\Delta, |v - V(s)|)$
6: **end for**
7: **until** $\Delta \geq \theta$
8: stable=true
9: **for** $s \in \mathcal{S}$ **do**
10: **if** $\pi(s) \neq \arg\max_a \sum_{s',r} p(s',r|s,\pi(s))[r+\gamma \cdot V(s')]$ **then**
11: $\pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,\pi(s))[r+\gamma \cdot V(s')]$
12: stable=false
13: **end if**
14: **end for**
15: **if** stable **then**
16: **return** V, π
17: **else**
18: go to line 2
19: **end if**

After a policy π is improved using $V_\pi(s)$ which yields a better policy π', we can then improve it again using $V_{\pi'}(s)$ to yield π''. In this way, we will obtain a sequence of monotonically improving policies and value functions.

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} v_{\pi_*}$$

Here E represents a policy evaluation whereas I represents a policy iteration. Since a finite MDP has a finite number of policies, this process must converge in a finite number of iterations to an optimal policy and optimal value function.

4.4.3 Greedy Iterative Augmentation

After deriving the optimal policies, we need to evaluate its performance. Here we used the expected cumulative reward(ECR) metric that computes the average of expected return under policy π from initial state s_0

$$ECR = \mathbb{E}_{s_0,\pi}[Q(s_0, \pi(s_0))]$$

The initial state is set the same for all students since we do not know any information about them prior to Ednet. Now we can use the greedy iterative augmentation algorithm to select the optimum feature representation[14]. The details are shown in Algorithm 3.

Algorithm 3 Greedy Iterative Feature Augmentation

Input: Feature space Ω, Action set $\mathcal{A}$, number of features N

1: Initialize feature representation $\hat{\Omega}$ and its corresponding state set $\mathcal{S}$
2: **for** i=1,2,... n **do**
3: **for** $\omega \in \Omega$ **do**
4: Create $\hat{\mathcal{S}}$ by adding feature ω into $\mathcal{S}$
5: π =Policy_Iteration($\hat{\mathcal{S}}, \mathcal{A}, \theta$)
6: $ECR = \mathbb{E}_{s_0,\pi}[Q(s_0, \pi(s_0))]$
7: **end for**
8: Add ω with max ECR into $\hat{\Omega}$
9: **end for**
10: **Return:** $\hat{\Omega}$

Note that the for-loop in line 2 of Algorithm 3 does not need to start with $i = 1$. In fact, since we already have three basic features, the size of $\hat{\Omega}$ initialized is 3, and we will start with $i = 3$. We ran the algorithm to obtain up to 5 extra features and the results are shown in Table 4.2.

Table 4.2: MDP and ECR improvement for each iteration

Iteration	MDP	ECR	Feature Space Size
base	MDP_lp	238.44	64
1	MDP_aug4	283.63	256
2	MDP_aug46	386.34	2048
3	MDP_aug468	392.59	4096
4	MDP_aug4687	393.97	16384
5	MDP_aug46873	394.02	65536

The value of 'Feature Space Size' is calculated by multiplying the bin size for each feature. As can be seen from the table, after 5 iterations, the increase in ECR becomes negligible and the algorithm converges. For the sake of simplicity, we only choose MDP_aug4687 as the optimal feature representation. The corresponding features for each MDP are shown in Table 4.3.

Table 4.3: Features included in each MDP

MDP	Features
MDP_lp	topic_fam,correct_so_far,avg_time
MDP_aug4	topic_fam,correct_so_far,avg_time,expl_received
MDP_aug46	topic_fam,correct_so_far,avg_time,expl_received,ssl
MDP_aug468	topic_fam,correct_so_far,avg_time, expl_received,ssl,prev_correct
MDP_aug4687	topic_fam,correct_so_far,avg_time,expl_received, ssl,prev_correct,av_fam

4.4.4 Monte Carlo Policy Evaluation

We chose MC methods to evaluate the model-based policies. For a given policy π, we generate an episode using π from a start state s_0. Since our derived MDPs do not have a terminating state, the episode could go on forever. To address this issue, we manually set the episode length to 1000. After generating the episode, we calculate the discounted reward at each state and sum up those rewards throughout the process. We run a total of 100 simulations and take the average of the cumulative rewards in the end. We also include the behaviour policy as a baseline for comparison. The technique used to carry out the actions observed in the dataset is referred to as the behaviour policy. Although it is usually unknown, it is critical that the behaviour policy's action choices are sufficiently diversified to guarantee a balanced distribution of support.The details are given in equation 4.1.

$$V_\pi = \frac{1}{100} \sum_{i=1}^{100} \sum_{t=1}^{1000} r \cdot \gamma^t \tag{4.1}$$

The results are shown in Figure 4.4.

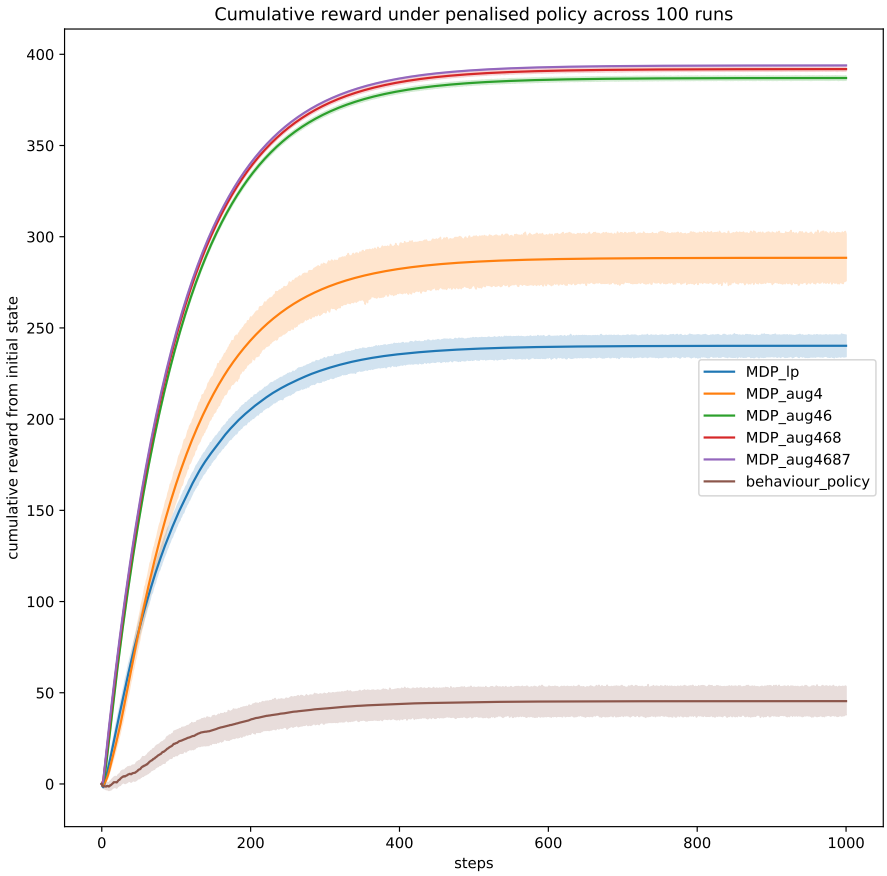

Figure 4.8: Cumulative discounted return from start state s_0

As can be seen from the graph, the cumulative rewards for most of the policies start to diminish after 200 steps. All the policies outperform the behaviour policy. The policies MDP_46,MDP_468,MDP_4687 have the highest cumulative rewards with minimal difference between them.

4.5 Model-Free Offline RL

4.5.1 Conservative Q-Learning

For large-scale real-world applications, effectively exploiting big, previously acquired datasets in reinforcement learning is a big challenge. Offline RL algorithms promise to learn effective rules using static datasets that have already been gathered without the need for additional intervention. Conservative Q-Learning[8] is a model-free offline RL method that builds upon the model-free online Q-Learning algorithm[18]. Q-learning is an off-policy reinforcement learning algorithm that seeks to find the best action to take

given the current state by minimizing the loss δ.

$$\delta = [Q(s_t, a_t) - (r_t + \gamma \cdot \max_{a_t \in A} Q(s'_t, a'_t))]^2$$

Here $Q : S \times A \rightarrow \mathbb{R}$ is a function that calculates the quality of a state–action combination. However, such methods face the challenge of distributional shift: learning effective skills requires deviating from the behavior in the dataset and making counterfactual predictions about unseen outcomes. The actions generated by the learning policy may be out of distribution to what is in the dataset, and the real Q-value of (s', a') may never be encountered. CQL makes an improvement by adding a regularization term onto the Q-Learning loss.

$$\delta_{CQL} = \delta + \alpha(\mathbb{E}_{s \sim \mathcal{D}, a \sim \mu}[Q(s,a)] - \mathbb{E}_{s,a \sim \mathcal{D}}([Q(s,a)]))$$

As a result, the function will minimize Q-values on unseen actions while maximizing the expected Q-value on the dataset at the same time. The policy would tend to stick with more familiar actions and the Q-value estimate is said to be more conservative.

To achieve CQL algorithm, we employ a open source offline RL library D3RLPY[17]. It is designed to implement the faster and more efficient training algorithms. Many state-of-the-art algorithms are available through d3rlpy's straightforward APIs: one can become a RL engineer without knowing deep learning libraries. In addition, d3rlpy is compatible with scikit-learn API, which is very popular and useful across the field of machine learning. Several parameters were tried in the experiment, including the hyperparameter α in equation 8, the Q-functions used, the dropout rate[15] and whether to perform batch normalization[6]. There are three types of Q-functions used. Besides the standard Q-function, there are also Quantile Regression(qr)[4] and Implicit Quantile Network(iqn)[3]. Instead of a single expected return, these two methods return a distribution of returns to improve stablility and convergence when used with function estimations. We ran a total of 9 CQL experiments, the details of which can be found in appendix 6.1. For each experiment, we ran 10 epochs. The results are shown in Figure 4.5. Note that while in the previous work[1] not all experiments finished 10 epochs due to time limits, this time we managed to run all experiments for 10 epochs thanks to the partition **Teach-LongJobs** of teaching clusters in University of Edinburgh which has a time limit of 3 days+8 hours.

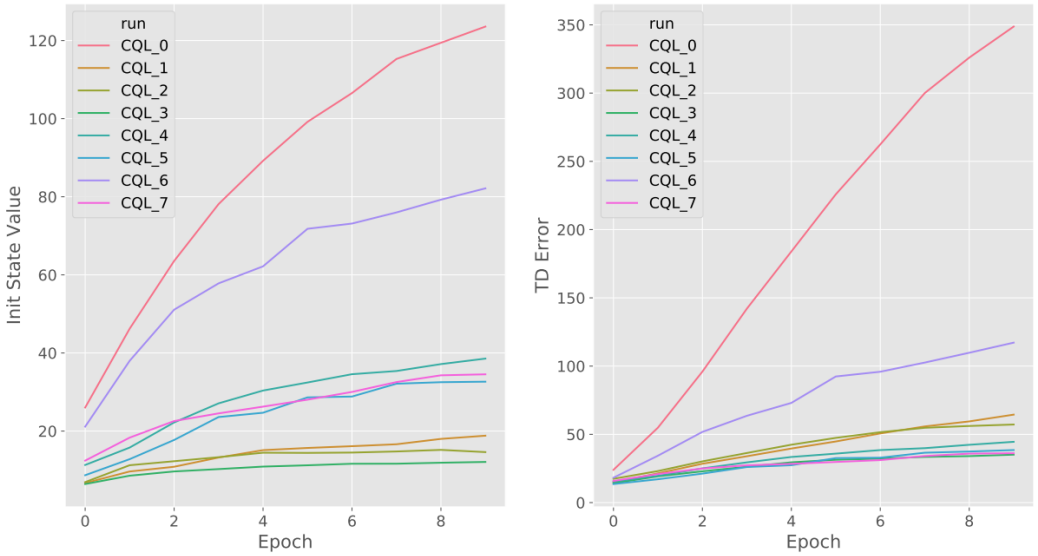

Figure 4.9: CQL Results. Left: ECR from initial state. Right: TD_error (Temporal Difference)

As can be seen from the graphs, although using the standard Q-function results in higher ECR, it also leads to higher TD_error, which suggests that it is prone to overfitting, hence not a suitable model for future prediction. In contrast, the policy 'CQL_7' that uses quantile regression with dropout rate 0.5, alpha-value 5 and batch normalization has a relatively high ECR and low TD_error, which is said to be more conservative. Comparing the graphs of 'CQL_4' and 'CQL_5', we can see that lowering α to 0.5 makes negligible difference. However, setting it to 0 will significantly increase both ECR and TD_error.

4.5.2 Fitted Q Evaluation

FQE is a model-free estimator that fits a Q-function under a specific policy based on observations in the training datasets. It approaches the evaluation as a supervised learning problem that makes use of function approximators[9]. The details are shown in Algorithm 2.

Note that the Q-values estimated are only dependent on the observed reward and value of the following state. This method is called bootstrapping. By doing so, FQE reduces compounded noise on value estimations across long episodes, resulting in smaller variance.

Algorithm 4 Fitted Off-Policy Evaluation with Function Approximation:FQE(π, c)

Input: Dataset $D = \{x_i, a_i, x_i', c_i\}_{i=1}^{n} \sim \pi_D$. Function class F. Policy π to be evaluated

1: Initialize $Q_0 \in F$ randomly
2: **for** k=1,2,...,K **do**
3: **for** i=1,2,...,n **do**
4: $y_i = c_i + \gamma Q_{k-1}(x_i', \pi(x_i'))$
5: **end for**
6: Build training set $\tilde{D}_k = \{(x_i, a_i), y_i\}_{i=1}^{n}$
7: Solve a supervised learning task:

$$Q_k = \arg\min_{f \in F} \frac{1}{n} \sum_{i=1}^{n} (f(x_i, a_i) = y_i)^2$$

8: **end for**
9: **Output:** $\forall x, \hat{C}^{\pi}(x) = Q_K(x, \pi(x))$

Similar to implementing CQL, we again use the library D3RLPY. The function estimator is a neural network with 2 layers, 256 hidden units and ReLU activations. There are two separate neural nets initilized: the value net and the target net. A value net is trained at each iteration when minimizing the loss in the equation between line 7-8 in Algorithm 2. A target net is trained at specific time intervals by making a hard-copy of the value net. The FQE loss in this context is

$$\delta(\theta) = \mathbb{E}_{s,a,s',r \sim \mathcal{D}}[(Q_\theta(s,a) - (r + \gamma \cdot Q_{\theta'}(s', \pi_e(s'))))^2] \tag{9}$$

where θ represents value net parameters and θ' represents target net parameters. We can use the function approximators to predict the value of the initial state under a specified CQL policy after training the network. This value can then be compared between various policies. The results are shown in Figure 4.6.

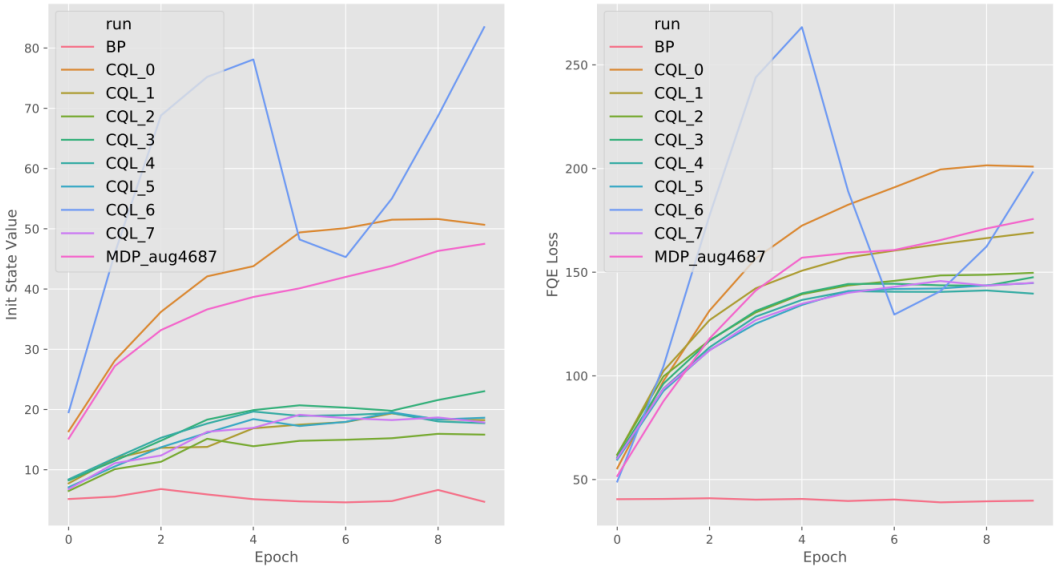

Figure 4.10: Left: FQE results on initial state values, $Q(s_0, \pi(s_0))$. Right: FQE Loss

The estimated behaviour policy (BP) is included as the baseline for comparison and it was outperformed by the various CQL policies as expected. 'CQL_6' yields the highest initial state value, but its loss is also the highest, which creates more uncertainty predicting $Q(s_0, \pi(s_0))$. Moreover, it is very unstable across epochs. The policy derived from 'MDP_aug4687' has a significantly higher initial state value than most of the CQL policies while the difference in losses is smaller, especially during the first two epochs.

Chapter 5

Extension

Some extension studies were done further to Aqil's work[1]. The extension consists of two aspects: perturbation on intermediate students and analysis of type of questions given to students.

5.1 Perturbation

In Aqil's work, perturbation on strong and weak students are performed to test the robustness of original policies. However, having only two categories is too broad to classify students accurately. This time, we made some adjustments to the original filter Ψ so that it also includes intermediate students. The new filter Ψ is shown in Table 5.1.

Table 5.1: Domain perturbation filters,Ψ for the three basic features

Feature	Strong	Intermediate	Weak
topic_fam	$\omega'_s > \omega_s, \omega'_s = 4$	$\omega'_s > \omega_s, \omega'_s \leq 3$	$\omega'_s = \omega_s$
correct_so_far	$\omega'_s \geq \omega_s, \omega'_s = 4$	$\omega'_s \geq \omega_s, \omega'_s \leq 3$	$\omega'_s < \omega_s$
avg_time	$\omega'_s \leq \omega_s, \omega'_s <= 2$	$\omega'_s \leq \omega_s, \omega'_s > 2$	$\omega'_s > \omega_s$

Here ω'_s represents the quantized next state's value while ω_s represents the current state's value. For strong and intermediate students, we would expect them to be more familiar to the topic after some learning engagement while for weak students, the learning effect may be so small that their familiarity would very likely stay the same. In addition, strong students are more likely to give a correct response and spend less time as they progress with learning while weak student will not perform as well. Intermediate students are obtained by further classifying the bin number of strong students. Strong students will have **topic_fam** and **correct_so_far** equal to 4, suggesting that they are extremely quick learner.

Next, we input *Psi* into Algorithm 2 in which transitions that satisfy this filter will be boosted by the constant c. This will increase the probability mass of this transition and affect the original MDP to make such transitions more likely.

Algorithm 5 Domain informed perturbations

Input: Set of features to perturb Ω , MDP transition probabilities $\mathcal{P}_{MDP}$, set of domain filters for each feature Ψ, positive perturbation constant $c = 0:05$

1: **for** $p_{s,a,s'} \in \mathcal{P}_{MDP}$ **do**

2: $\quad \Delta_{s,a,s'} = p_{s,a,s'} + \sum_{\omega \in \Omega} \Delta_\omega$ Where $\Delta_\omega = \begin{cases} 0.05 & \omega'_s, \omega_s satisfy \Psi_\omega \\ 0 & \text{otherwise} \end{cases}$

3: **end for**

4: Adjust $\Delta_{s,a,s}$ relative to others within the s,a pair

5: $\Delta_{s,a,s} = \Delta_{s,a,s} + \frac{1}{|\Psi|} \sum_{s'} \Delta_{s,a,s'}$

6: Set perturbed transition probabilities $\bar{\mathcal{P}}_{MDP} = \mathcal{P}_{MDP}$

7: **for** $p_{s,a,s'} \in \bar{\mathcal{P}}_{MDP}$ **do**

8: $\quad p_{s,a,s'} = max(p_{s,a,s'} + \Delta_{s,a,s'}, 0)$

9: **end for**

10: **for** $p_{s,a,s'} \in \bar{\mathcal{P}}_{MDP}$ **do**

11: $\quad p_{s,a,s'} = \frac{p_{s,a,s'}}{\sum_{s'} p_{s,a,s'}}$

12: **end for**

13: **Return:** $\bar{\mathcal{P}}_{MDP}$

The cumulative reward from initial states across 1000 steps for various MDPs is shown in Figure 4.7

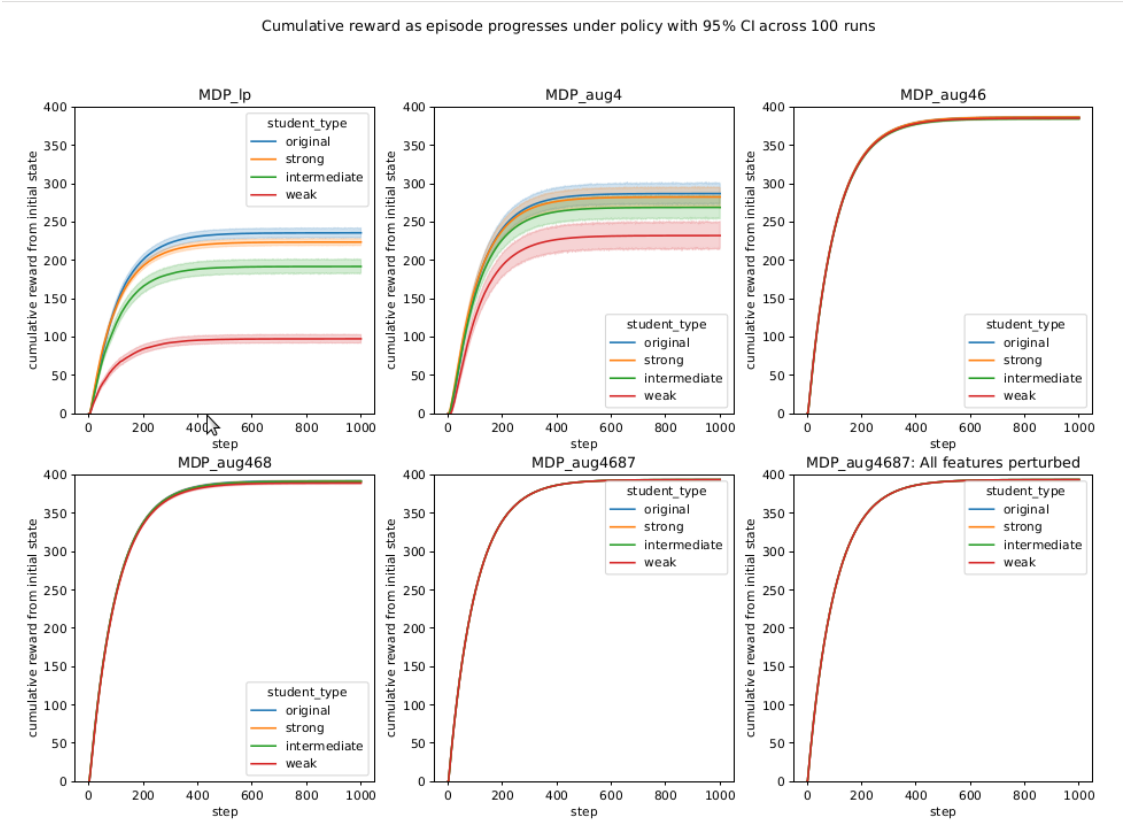

Figure 5.1: MC Policy Evaluation of the original policy under the perturbed MDPs

From the graphs, we can see that the original MDP always has the best performance because it is where the policy is derived. When only perturbing three features, weaker students are affected more. As shown by the top-left graph, there is a big difference between reward of weak students and reward of intermediate students. However, as more features are included, the effect starts to diminish. The bottom-right graph shows that even with all features perturbed, the performance of policy is not affected. This means that the policies would be robust in the real-world setting where students have different learning styles. One drawback is that including more features inevitably increases the cost of computation. A policy that favors weaker students would be desired.

5.2 Policy Analysis

In this section, we will zoom into the details of various MDPs created and analyze the policies given to each state. Again, we classify students into three groups and count the number of questions given to them in each level. The classification criteria is given in Table 5.2.

Table 5.2: Conditions for classification into strong,intermediate and weak students.

Strong	Intermediate	Weak
$(\omega_t + \omega_c) > 6$, $\omega_a = 1$	$4 \leq (\omega_t + \omega_c) \leq 6$, $\omega_a = 2$	$(\omega_t + \omega_c) \leq 3$, $\omega_a \geq 3$

Here $\omega_t, \omega_c, \omega_a$ denote the bin number of the feature **topic_fam,correct_so_far,avg_time** respectively. Since larger values of ω_t and ω_c indicate stronger performance while larger value of ω_a indicates weaker performance, we use separate conditions for ω_a. After the classification, we find the number of questions/lectures given to each student group for each difficulty level. The results for MDP_aug4687 are shown in Table 5.3-5.5.

Table 5.3: Number of questions/lectures offered to strong students

Level	0	1	2	3	4
Number offered	26	32	47	78	155
Percentage	7.69	9.47	13.90	23.08	45.86

Table 5.4: Number of questions/lectures offered to intermediate students

Level	0	1	2	3	4
Number offered	209	218	186	392	593
Percentage	13.08	13.64	11.64	24.53	37.11

Table 5.5: Number of questions/lectures offered to weak students

Level	0	1	2	3	4
Number offered	262	176	185	290	343
Percentage	20.86	14.01	14.73	23.09	27.31

It is clear from the table that strong students get the highest proportion of difficult questions (Level 4) and weak students get a much smaller proportion. Instead, weak students are advised to watch more lectures (Level 0) than strong and intermediate students. However, more questions in Level 3 and Level 4 are given to students across all three groups, which is undesirable for weak students as they would need more questions from Level 1 and Level 2. This could be due to the fact that the data is more biased towards hard questions(see 4.6 in section 4.3.2). Nonetheless, weak students still get much less Level 4 questions. The proportions of Level 2 and Level 3 questions offered are similar across all three groups. Results for other MDPs can be found in Table 7.7-7.10 in the Appendix.

Chapter 6

Conclusion

In this project, we introduced two educational contents that help students improve their learning with the technique of machine learning and artificial intelligence. The first one is a difficulty ranking algorithm called EduRank that gives personalized recommendation of a series of questions to students. Instead of using accuracy, which can be rigid in the context of question ranking, we used AP score as the metric to evaluate performance of the algorithm. The algorithm was tested on two datasets, PSLC and Ednet, and has AP score of 0.57 and 0.54 respectively. The second one is a reinforcement learning agent that derives learning policies using both the model-based MDP and the model-free CQL method. For the model-based approach, seven features are selected for the optimal state representation with an ECR of around 394. For the model-free approach, FQE are run on 8 CQL policies and 'CQL_0' stands out from the rest. In addition, the experiment of perturbation was performed on strong, intermediate and weak students. The result shows that weak students are more susceptible to perturbation of the model while including sufficient number of features will greatly reduce the impact of perturbation.

Several challenges were encountered throughout the project. Initially when running EduRank on PSLC dataset, we used extra iterations that makes the calculation task unnecessarily heavy. After analysing the bottleneck complexities of the algorithm, we identified those problems and cached relevant results before entering another iteration. In the end, the run time of the algorithm is considered reasonable due to the limitation of python. The next challenge is to run CQL and FQE during the RL stage. Normally the training process takes days to run even with the help of cluster computing. As a result, the cost of making mistakes in the input arguments becomes extremely high. There have been many times that after days of training, we found a mistake in the input arguments and the whole training became irrelevant. Much patience is required for such tasks.

We also found several limitations of EduRank and Ednet. The biggest limitation is that the data (both PSLC and Ednet) are not collected in a closed environment. Although Ednet captures some user actions that could impact correctness of reponse to questions, students can seek help from other ways. For example, if a students ask his classmates for the answer to a question, then this action will not be recorded by the system. To simulate

a real-life classroom setting more closely, pre/post test records are needed so that the results can reflect students' strength more accurately. Another aspect of improvement would be to make both EduRank and the RL agent focus more on weak students. Giving boring questions to strong students would hurt much less than giving difficulty questions to weak students since strong students are more adaptable. Moreover, there is much larger room for improvement from weak students. Hence, in the future, we may zoom in to weak students and construct corresponding learning models for them. In addition, when quantizing questions into four bins, we only used the percentage correctness as the metric. A more rigorous way would be to use a similar tie break method in EduRank by time taken to solve the questions.

Bibliography

[1] Zainal Aqil Zainal Azhar. Designing an offline reinforcement learning based pedagogical agent with a large scale educational dataset. 2021.

[2] Youngduck Choi, Youngnam Lee, Dongmin Shin, Junghyun Cho, Seoyon Park, Seewoo Lee, Jineon Baek, Chan Bae, Byungsoo Kim, and Jaewe Heo. Ednet: A large-scale hierarchical dataset in education. In *International Conference on Artificial Intelligence in Education*, pages 69–73. Springer, 2020.

[3] Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, pages 1096–1105. PMLR, 2018.

[4] Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[5] Yossi Ben David, Avi Segal, and Ya'akov Gal. Sequencing educational content in classrooms using bayesian knowledge tracing. In *Proceedings of the sixth international conference on Learning Analytics & Knowledge*, pages 354–363, 2016.

[6] Sergey Ioffe and Christian Szegedy Batch Normalization. Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2014.

[7] Kenneth R Koedinger, Ryan SJd Baker, Kyle Cunningham, Alida Skogsholm, Brett Leber, and John Stamper. A data repository for the edm community: The pslc datashop. *Handbook of educational data mining*, 43:43–56, 2010.

[8] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[9] Hoang Le, Cameron Voloshin, and Yisong Yue. Batch policy learning under constraints. In *International Conference on Machine Learning*, pages 3703–3712. PMLR, 2019.

[10] Seewoo Lee. Ednet, 2020. URL:`https://github.com/riiid/ednet`.

[11] 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.

[12] Avi Segal, Kobi Gal, Guy Shani, and Bracha Shapira. A difficulty ranking approach to personalization in e-learning. *International Journal of Human-Computer Studies*, 130:261–272, 2019.

[13] Avi Segal, Ziv Katzir, Kobi Gal, Guy Shani, and Bracha Shapira. Edurank: A collaborative filtering approach to personalization in e-learning. In *Educational Data Mining 2014*. Citeseer, 2014.

[14] Shitian Shen and Min Chi. Aim low: Correlation-based feature selection for model-based reinforcement learning. *International Educational Data Mining Society*, 2016.

[15] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[16] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[17] Michita Imai Takuma Seno. d3rlpy: An offline deep reinforcement library. In *NeurIPS 2021 Offline Reinforcement Learning Workshop*, December 2021.

[18] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[19] Emine Yilmaz, Javed A Aslam, and Stephen Robertson. A new rank correlation coefficient for information retrieval. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 587–594, 2008.

Chapter 7

Appendix

7.1 CQL parameters

Table 7.1: Model name and corresponding parameters for each experiment.

Model Name	Q-function	batch_normalization	dropout rate	alpha
CQL_0	standard	No	No	1
CQL_1	qr	No	No	1
CQL_2	iqn	No	No	1
CQL_3	qr	No	0.5	1
CQL_4	qr	Yes	0.5	1
CQL_5	qr	Yes	0.5	0.5
CQL_6	qr	Yes	0.5	0
CQL_7	qr	Yes	0.5	5
bcq_qr_bndo	qr	Yes	0.5	1

7.2 KT2

timestamp	action_type	item_id	source	user_answer	platform
1358114668713	enter	b4957	diagnosis		mobile
1358114691713	respond	q6425	diagnosis	c	mobile
1358114701104	respond	q6425	diagnosis	d	mobile
1358114712364	submit	b4957	diagnosis		mobile
1358114729868	enter	b5180	sprint		mobile
1358114745592	respond	q6815	sprint	c	mobile
1358114748023	respond	q6816	sprint	a	mobile
1358114748781	respond	q6814	sprint	a	mobile
1358114751032	submit	b5180	sprint		mobile

Figure 7.1: Example student data in KT2. The student solved 4 questions on mobile, 3 of which were contained in the bundle b5180.

7.3 KT3

timestamp	action_type	item_id	source	user_answer	platform
1573364188664	enter	b790	sprint		mobile
1573364206572	respond	q790	sprint	b	mobile
1573364209673	respond	q790	sprint	d	mobile
1573364209710	submit	b790	sprint		mobile
1573364209745	enter	e790	sprint		mobile
1573364218306	quit	e790	sprint		mobile
1573364391205	enter	l540	adaptive_offer		mobile
1573364686796	quit	l540	adaptive_offer		mobile
1573364693793	enter	b6191	adaptive_offer		mobile
1573364702213	respond	q8840	adaptive_offer	c	mobile
1573364705838	submit	b6191	adaptive_offer		mobile

Figure 7.2: Example student data in KT3. After the student solved question q790, they read the explanation associated with q790 and watched lecture l540.

7.4 KT4

timestamp	action_type	item_id	cursor_time	source	user_answer	platform
1358114668713	pay	p25				mobile
1358114691713	enter	b878		sprint		mobile
1358114701104	text_enter	q878		sprint		mobile
1358114712364	play_audio	q878	0	sprint		mobile
1358114729868	pause_audio	q878	10000	sprint		mobile
1358114745592	eliminate_choice	q878		sprint	a	mobile
1358114748023	respond	q878		sprint	c	mobile
1358114748781	submit	b878		sprint		mobile
1358114751032	enter	e878		sprint		mobile
1358114779211	play_audio	e878	0	sprint		mobile
1358114792300	pause_audio	e878	8000	sprint		mobile
1358114842195	quit	e878		sprint		mobile

Figure 7.3: Example student data in EdNet-KT4. After the student bought an item, they solved the LC question q878. The timestamps at which they played and paused audio were recorded. They also eliminated ‘a’ and chose ‘c’ as an answer.

7.5 Questions

question_id	bundle_id	explanation_id	correct_answer	part	tags	deployed_at
q2319	b1707	e1707	a	3	179;53;183;184	1571279008033
q2320	b1707	e1707	d	3	52;183;184	1571279009205
q2321	b1707	e1707	d	3	52;183;184	1571279010285
q2322	b1708	e1708	b	3	52;183;184	1571279012823
q2323	b1708	e1708	c	3	179;52;182;184	1571279013890
q2324	b1708	e1708	d	3	52;183;184	1571279014989

Figure 7.4: Example questions in EdNet. Questions are formed in bundles, with skill tags, part and correct answer provided.

7.6 Lectures

lecture_id	part	tags	video_length	deployed_at
l805	5	99	230000	1570426256512
l855	0	203	253000	1570425118345
l1259	1	222	359000	1570424729123
l1260	1	220	487000	1570424738105
l1261	1	221	441000	1570424743162
l1262	1	223	587000	1570424748780

Figure 7.5: Example lectures in EdNet, with skill tags, part and video length provided.

7.7 MDP_lp

Table 7.2: Number of questions/lectures offered to students in MDP_lp.

Level	Strong	Intermediate	Weak
0	0	6	2
1	0	2	0
2	0	0	0
3	0	2	0
4	3	0	4

7.8 MDP_aug4

Table 7.3: Number of questions/lectures offered to students in MDP_aug4.

Level	Strong	Intermediate	Weak
0	1	4	1
1	1	7	5
2	0	3	6
3	2	14	7
4	6	8	5

7.9 MDP_aug46

Table 7.4: Number of questions/lectures offered to students in MDP_aug46.

Level	Strong	Intermediate	Weak
0	8	39	46
1	13	52	44
2	5	36	29
3	10	56	32
4	19	77	41

7.10 MDP_aug468

Table 7.5: Number of questions/lectures offered to students in MDP_aug468.

Level	Strong	Intermediate	Weak
0	19	162	209
1	29	122	79
2	16	73	65
3	23	139	104
4	61	238	118