Designing an Offline Reinforcement Learning Based Pedagogical Agent with a Large Scale Educational Dataset

Zainal Aqil Zainal Azhar

Master of Science

Data Science

School of Informatics

University of Edinburgh

2021

Abstract

As e-learning platforms continue to develop, a new trend of intelligent tutoring systems begin to emerge in the form of adaptive pedagogical agents. These AI driven platforms enable a more individualised learning experience that adapts to the user's evolving pedagogical needs throughout a course. In this report we create a Reinforcement Learning (RL) based pedagogical agent that is able to optimize the sequencing of learning materials to maximize learning as measured by expected future student performance. We conduct the training completely offline based on the publicly available dataset EdNet, a large-scale hierarchical dataset of diverse student activities collected by an online learning platform, *Santa* [9]. We confront the challenges of offline RL through the construction of a student model in the form of a Markov Decision Process derived from observations in the dataset. A feature pool is created from the raw logs in EdNet, from which, different state representations are formed by sampling greedily using an iterative augmentation procedure. Our model-based policies were trained using Policy Iteration and evaluated using the Expected Cumulative Rewards (ECR), or the average of the expected return at the initial state under the policy.

Our goal was to explore the influence of the state representations on the performance of the RL agent and its robustness towards perturbations on the environment dynamics. We show that a larger, more complex representation constituting of more features, yields better policy performance. We also show that the policies derived from the larger representations are more robust towards perturbations in the expected environment induced by students with different learning characteristics i.e. stronger and weaker learners. We then explore an alternative model-free offline RL approach utilising the Conservative Q-learning algorithm. Performance comparisons between the model-based and model-free approaches were made using two Offline Policy Evaluation techniques, Importance Sampling and Fitted Q Evaluations. The behaviour policy was estimated as a reference baseline. We demonstrate that at least one of the two methods outperformed the baseline in most metrics.

Acknowledgements

It has been a challenging few months, but I have made it through thanks to those who support me professionally and personally. I would like to thank my family and friends for always lending an ear to my endless complaints and troubles. Whenever my script crashes or results disappoint, they would be the first to hear.

And of course I have to thank my supervisors, Avi Segal and Kobi Gal for their continuous support and ideas throughout the project. Besides, providing technical help, their words of encouragement and supportive tone really gave me the confidence to explore this otherwise new field for me.

Finally, I also want to dedicate this project to my late grandparents. I wish you can be here to celebrate this moment with me.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

E-learning platforms have seen a surge in popularity over the last decade [19], spurred on by the increased internet penetration into developing communities [6]. The target demographic has expanded beyond casual users/students as more organizations adopt e-learning to train their workforce and actively engage them in life-long learning [61]. E-learning platforms come with intrinsic advantages over traditional learning. These include its high accessibility and its on-demand nature making it a more convenient experience relative to its face-to-face counterpart [39]. While e-learning (and its different forms) has been omnipresent since the mid-late 20th century, the concept of 'intelligent' learning platforms has recently emerged as a new trend within this domain [26].

These platforms are driven by expansion of research into applied artificial intelligence (AI) and have been addressed in literature under various names such as 'Intelligent Tutoring Systems' (ITS) [11] or 'Pedagogical Agents' [58]. The common objective is to create an adaptive learning environment, with an emphasis on 'generativity' or the ability to generate customized problems, hints, or support for each individual student [61]. Traditional self-paced online courses, such as those used in MOOCs (Massive Open Online Courses), create a static learning path often without contextualized feedback or personal adaptive guidance. In such scenarios, a user might feel disinclined to progress through the course material if they encounter difficulties in following the generic course structure. This will often lead to them losing interest and disengage from the course by dropping out, as is so often observed in distance education [61]. Therefore, there is a growing need for a personalized sequencing of contents/support that can adapt to the individual differences of the student as well as their

evolving pedagogical requirement throughout the course progression [49]. Intelligent tutoring systems, pedagogical agents and the like, are the proposed solution within the applied AI in education community. With large educational datasets becoming more prevalent[9][44], these AI-driven solutions become more feasible.

1.2 Problem Statement

Research in cognitive science has long demonstrated the strong correlation between topic sequencing and learning [47]. Static e-learning platforms lack the capacity to respond to a student's 'cognitive state' and therefore perform poorly relative to a human tutor [61]. A students' cognitive state is a term used to describe an explicit representation of their learning characteristics [28]. It can be composed of multiple features such as a student's general competency or aptitude, learning style and confidence level [61]. One of the main difficulties in constructing an intelligent learning environment, is approximating this cognitive state since it is not directly observable [17]. However, a more achievable goal is to model this state to a degree where an agent can leverage the information to provide effective sequencing [61]. This model can be constructed in a variety of ways and is commonly referred to as the 'student model' in literature [28][61]. The fidelity of the model could significantly impact the agent's performance since a more complex representation of the cognitive state will provide more information to the pedagogical agent [61].

While constructing a model is usually the first challenge, another that soon follows is the need for an optimization algorithm to utilise the model in enhancing learning. Reinforcement learning (RL) provides the perfect mathematical machinery to optimize a learning sequence towards this predefined objective [17]. RL is a class of machine learning algorithms, that optimize a control problem where an 'agent', learns an optimal 'policy' that maps 'states' to a given 'action' in such a way as to maximized a 'reward' function [55]. One can quickly observe the relevance of such algorithms in the context of ITS. Here, a learning sequence is optimized based on a numerical reward (for example test marks) by a pedagogical agent that prescribes actions (adaptive feedback/sequence) based on different states (approximations of the cognitive states).

Although RL directly addresses our goal oriented problem, its major shortcoming lies with its requirement for 'online' training [30]. Online training refers to an interleaved process of training (or deriving a policy) and collecting more data, where the data collection distribution depends on the current policy. This cycle occurs repeatedly

for many iterations. In many applications (including this project), online training is not feasible due to its expensive and potentially dangerous implications [35]. One can imagine the severe costs of an untrained RL agent interacting in an autonomous vehicle setting to collect more data. This online procedure is clearly different than the highly successful supervised learning setting where the data collection process only occurs **once** before training commences [35]. The ongoing efforts in redesigning RL to leverage big data, has led to a new field of research known as offline RL [43][22][34]. In offline RL, the learning is performed only on a fixed dataset that was collected once before training. As we will later explore, different algorithmic modifications are proposed to correct for estimation errors that result from inability to train online.

1.3 Objectives

In this project, we are developing an adaptive RL based pedagogical agent that is able to optimize the sequence of learning materials (questions and lectures) to maximize student performance as measured by the expected ability to answer questions correctly at varying levels of difficulties. This agent will have the capacity to respond to a student's current state as they progress through the learning material. More importantly the agent will be trained completely offline using the EdNet dataset, an extensive collection of student logs from an online learning platform, *Santa* [9]. To address the challenges of offline RL, we will first create a model in the form of a data-driven Markov Decision Process (MDP) and utilise a Dynamic Programming based algorithm, Policy Iteration, to derive an optimal policy. We will then explore a more direct model-free approach and compare the performance. Our objective can be broken down into several components:

- Investigate the impact of the state representations on the agent's performance and choose an optimal representation accordingly. To achieve this, we will begin with a base representation consisting of simple features to form a base MDP model. Policy Iterations uses the model to obtain an optimal policy, which can subsequently be evaluated. A greedy iterative augmentation procedure (visualized in figure 1.1) is then employed to augment the representation space, by incrementally adding new features and choosing the best performing representations greedily.

- Analyse the robustness of the policies from the different representations against

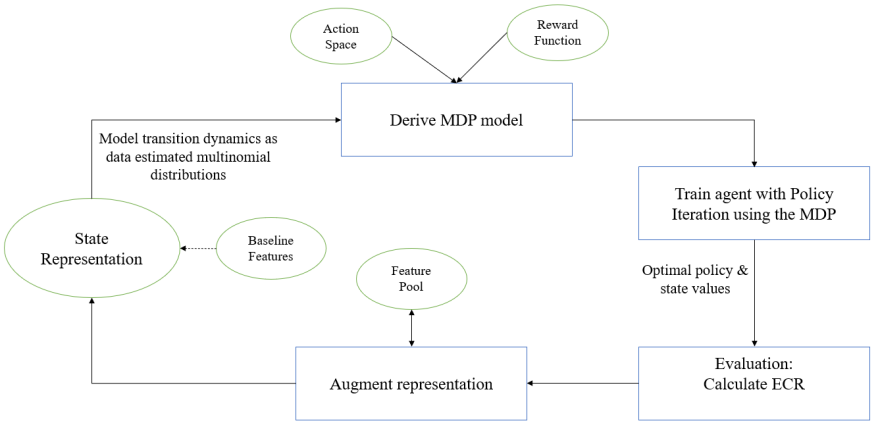


Figure 1.1: Greedy iterative feature augmentation pipeline

perturbations in the model dynamics corresponding to students with varying learning characteristics.

- Explore an alternative model-free approach to compare with the model-based policies and a baseline behaviour policy (data gathering policy) under several Offline Policy Evaluation (OPE) techniques. These techniques estimate real-world performance using only the collected dataset i.e. without empirical testing.

We show that a larger, more complex representation constituting of more features, yields better policy performance and is more robust towards perturbations in the model induced by students with different learning characteristics. We also demonstrate that at least one of the two methods (model-based and model-free) outperformed the behaviour policy in the OPE metrics.

1.4 Document Structure

The remainder of this dissertation is organized as follows. Chapter 2 will cover the background of Reinforcement Learning and Markov Decision Processes and their applications in the ITS domain. Chapter 3 will cover the derivation of the action space and reward function and the features used in forming the state representations. Chapter 4 will address the student model (data-driven MDP) construction, Policy Iteration training, greedy iterative augmentation pipeline and the model-free approach. Chapter 5 covers the different Offline Policy Evaluation techniques and finally, Chapter 6 will present all results and accompanying discussion.

Chapter 2

Background

2.1 Reinforcement Learning & Markov Decision Processes

Reinforcement Learning (RL) is a group of machine learning algorithms that deal with optimal control problems [55]. RL involves deriving an optimal strategy or policy that maximizes a numerical scalar 'reward'. The policy consist of a prescribed 'action' at each 'state' in the state space. Traditionally, training an RL agent involves repeated interactions with an environment, where at each time-step, the agent can observe the changes in the environment through the information constituting the 'state'. This is knows as online training. The agent interacts with the environment using an 'action' selected from a finite set of available actions. In doing so, the environment yields a numerical 'reward' and transitions into a following state, where the process repeats.

The RL framework is governed by a Markov Decision Process [55][34][8] that is defined by the tuple of $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S} = s_1, .., s_n$ defines the complete space of every possible state, $\mathcal{A} = a_1, .., a_n$ represents all available actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ designating the transition probabilities between states conditioned on an action and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ denotes the reward function that is conditioned on the state, action and observed next state. Figure 2.1 shows a depiction of the agent interacting with the environment at each time step t within the MDP framework.

The goal of the agent is to maximize the **cumulative** reward it accumulates from each state. This cumulative reward is usually discounted by a factor γ raised to the power of t, to represent a lower perceived value for rewards received further in the future. The policy is a mapping of optimal actions to each state in $\mathcal{S}$. The discounted

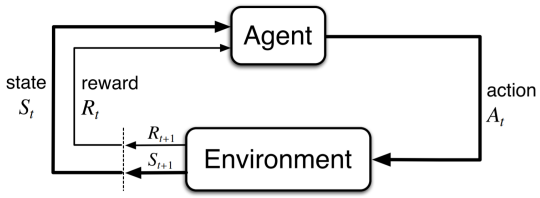


Figure 2.1: Visual depiction of the MDP framework [55]

cumulative reward is denoted as the 'return', and the return from a particular state is associated with a policy π and the transition dynamics $\mathcal{P}$. The reward function ties the agents optimization goal with the modeller's actual objective. Therefore its design must ensure those two criteria are aligned. Note that the policy can be stochastic where it prescribes a distribution of actions at each state. The return of a state (or the state value) under a policy π can be defined as $V(s) = \mathbb{E}_{a\sim\pi,r,s\sim\mathcal{P}}[\sum_{t=0}^{\inf}\gamma^t r(s_t,a_t)|s_0 = s]$.

In most cases, the transition dynamics $\mathcal{P}$ are not known. If interaction with the environment is possible, then this information is not needed since the samples it draws from the environment will follow the environment dynamics [55]. RL algorithms that are designed to perform this way are known as model-free. An important element for these algorithms is some mechanism to trade-off between 'exploration' and 'exploitation'. Exploration refers an agent's preference to take random actions to explore the unknown environment and learn about the expected returns from different states [55]. Exploitation describes the agent's preference to maximize its returns using its current information. Intuitively, most model-free RL algorithms are designed to be more explorative at the start and progressively taper to a more exploitative behaviour. During the exploration phase, it must be noted that a large amount of interaction is required for the agent to have accurate information on the environment [55][42].

In 'tabular' RL, a state value is updated only from the experiences that involve the state [55]. If a state is not encountered in the agent's past experiences, then nothing can be said about its value. 'Function Approximation' can help alleviate this problem, by generalizing a state's value based on features i.e. states with similar features will have similar values. As in supervised learning, parametrized function approximators learn weight vectors that approximate a state's value from its features. Even with function approximation, online model-free RL algorithms are generally appropriate for domains where environment interaction is computationally cheap and feasible [30].

The alternative to model-free RL is model-based RL. These approaches are more prevalent in domains where environment interactions are cost prohibitive [30][35].

Model-based RL requires a model that holds information regarding the environment dynamics i.e. the transition probabilities $\mathcal{P}$ and reward function $\mathcal{R}$. The model mimics the behaviour of an environment and allows inferences about how the environment will respond to actions [55]. In approaches where the model is driven by sample data from the environment, model-based RL tends to be more sample efficient than model-free RL [31][45]. However, an important caveat to model-based RL is that the real-world performance will be heavily influenced by the quality of the model [30].

2.2 Previous work in RL Based Pedagogical Agents

The majority of work in this domain involves creating some form of student model (or simulator) to utilise **in training the RL-agent** [17]. A common approach in prior work is to integrate learning/cognitive theory into the construction of the student model. This imparts domain knowledge into the workflow and is shown to yield positive results [17]. An example of such approach is the work by Bassen et al. [2]. Their objective was to optimize the sequencing of learning material from different knowledge components (KCs), to 'maximize learning'. Their reward function is based on the difference between a post-test score (taken by users after completing the course) and a pre-test score (taken before the course). This metric is denoted as the Normalized Learning Gain (NLG). Training the agent with human participants is far too resource intensive. Instead their training is performed on a 'simulated learner' based on Bayesian Knowledge Tracing (BKT), a cognitive model that aims to estimate a learner's mastery of different skills. The learner's response to a particular question can be simulated based on the mastery of the related skill. The parameters of the BKT were set based on domain knowledge. Segal et al. [49] also utilised a similar cognitive based model, Item Response Theory (IRT) [25] to simulate student responses to questions of different level of difficulty. This is especially relevant since their objective was to sequentially deliver questions of differing levels of difficulty (rather than KCs) to maximize learning gains.

Other approaches forgo the framework of established learning models and instead manually design their simulators further integrating domain expertise. Dorça et al. [16] employed a probabilistic model to simulate the learning process. Instead of sequencing activities by KCs or difficulties, they sequenced activities based on its associated learning style i.e. visual, verbal etc. Therefore, their simulator was manually designed based on research surrounding these principles. Iglesias et al. [28] utilised an expert

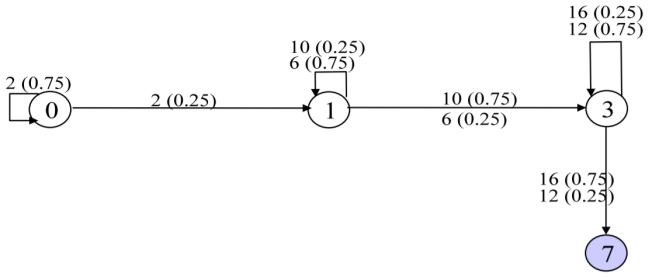


Figure 2.2: Expert derived MDP [28]. Actions and states (in circles) are encoded by integers and transition probabilities are provided in the parentheses.

derived artificial Markov Decision Process to act as the student model. This entails manually describing the state space, transition probabilities and rewards. A subset of the artificial MDP is shown in figure 2.2. Similar to previous student models, this MDP can be used to simulate student responses to train the RL agent.

The works described so far do not utilise historical data to derive their student model. While integrating expert knowledge can be beneficial, a completely data-free proposition could impart strong biases. In our implementation, we take an alternative approach in using a purely data-driven model. There are existing literature which also do the same. For example, the authors of [52][56][7][48] employed data-driven MDPs as their student model. Different than the handcrafted MDP in [28], the transition probabilities and reward functions in these MDPs were obtained from the aggregated statistics observed in the dataset, further elaborated in section 4.1.1. Data-driven student models in literature were not only limited to data-driven MDPs. For instance, Beck et al. [3] utilised a linear regression model denoted as Population Student Model (PSM). PSM was trained on student trace data from a learning software and could simulate time taken and probability of a correct response.

Data-driven simulators require a quality training corpus that is sufficiently large and varied [52][30]. In contrast to EdNet (a massive dataset collected over several years), the authors of these papers were limited to much smaller scale datasets that were collected from a single cohort. To the best of our knowledge, our implementation is the first in RL based pedagogical agents to utilise a large scale MOOC dataset.

Chapter 3

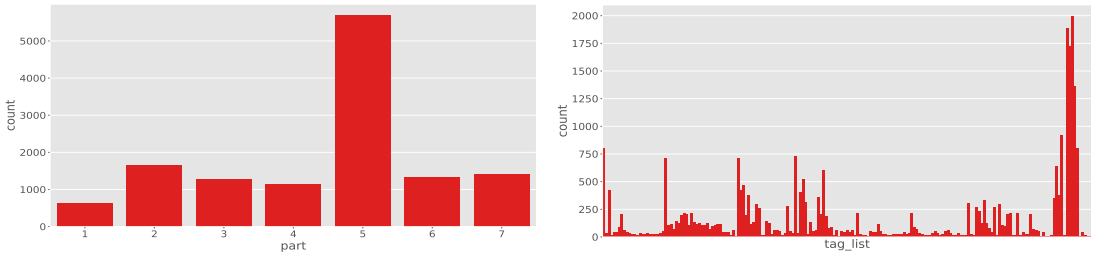
Exploration of EdNet & Feature Extraction

3.1 Data Description

EdNet is massive dataset of student logs from a MOOC learning platform in South Korea, *Santa*, collected by *Riiid! AI Research*[1]. Santa covers a preparation course for the TOEIC (Test of English for International Communication) English proficiency exam. There are a total of 131,441,538 interactions collected from 784,309 students on the e-learning platform. Each student's interaction log is stored on an individual csv (Comma-Separated Values) file. These consist of user records of questions attempted, lectures watched and explanations reviewed, along with other meta information. EdNet logs are presented at 4 levels of hierarchy with higher levels providing higher fidelity logs. These levels are denoted as KT1 - KT4. The most coarse hierarchy, KT1, has logs that capture only question-answer pairs and their elapsed times (see example in the appendix in table 8.1).

Progressing into the next fidelity level, KT2 logs now also show answer oscillations before the student submits a final answer on a question. KT3 includes logs of lectures watched and the explanations reviewed. These are recorded in real time to provide an accurate chronological record of the students' interaction with the platform. At highest hierarchy, KT4 records detailed actions such as playing/pausing lectures and payment related information.

[1] `https://www.riiid.co/`



(a) Parts (b) Skills

Figure 3.1: Unique question distribution in 'parts' vs 'skills' grouping

3.2 Data Exploration: Action Space

The main incentive in exploring the data is to observe potential candidates for the action space, state features and reward function. EdNet provides a total 13,169 questions and 1,021 lectures tagged with 293 types of skills [9]. Each question and lecture is segregated into one unique 'part', with 7 parts in total: part 1 to part 7. No specific information is provided on the part grouping criteria, however we assume that each part was grouped based on some meaningful domain criteria like the KCs in Bassen et al. [2]. Note some parts are limited to paying users on the platform [9]. The distribution of the total number of unique questions available in each part is shown in figure 3.1. Apart from part 5, every other part is fairly balanced in terms of questions available. EdNet offers a finer grouping of question/lectures according to the 'skills' (293 in total) they entail. Each question/lecture is tagged with 1 or more of these skills. Both options represent good candidates for the action space since they group the numerous questions/lectures in a domain meaningful manner. Simply assigning each unique activity (question/lecture) a discrete action in the action space is not feasible due to the enormous and diverse support it requires from the dataset (further explained in section 4.1.1). However, even the 293 unique classes observed in skills is far too granular. Additionally, as shown in figure 3.1, the question bank is more unevenly distributed with respect to the skills grouping, which can lead to difficulties in getting equal support (or supporting observations) for each class from the dataset. Therefore, 'part' was chosen on the basis that its 7 unique classes is more manageable.

However, this means that our pedagogical agent can only control which part to present to a user in its sequencing, and some other algorithm would decide which of the many possible activities within a part is actually chosen. A balance must be made between a manageable action space size and a useful pedagogical agent. Considering

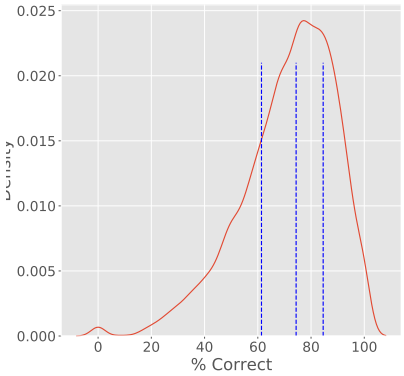
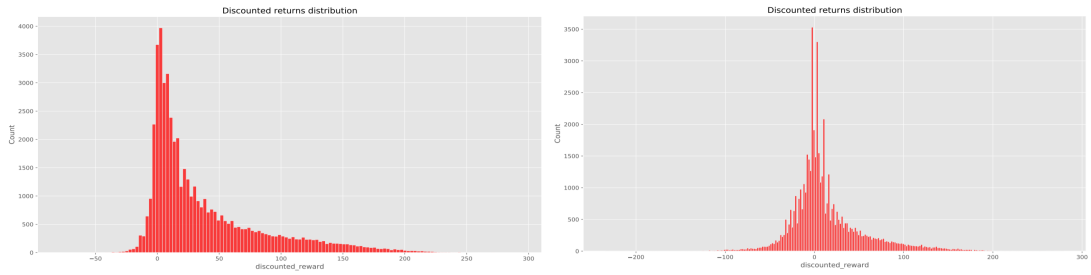


Figure 3.2: Distribution of question difficulty in the question bank. Blue lines are derived quantiles

the large support available in EdNet, we decided to extend the action space, discretizing the questions in terms of difficulty, similar to that in Segal et al. [49]. Now the agent can prescribe the part and also the level of difficulty of the activity. The difficulties was quantized into 4 levels ranging from 1-4. Unfortunately EdNet does not classify questions/lectures into difficulty levels, meaning that they would need to be inferred from the student logs. One way of achieving this is to measure the percentage of correct answers submitted for a question and compare it with other questions in the question bank. A distribution of question difficulties can be created with each item being a unique question in the question bank. Quantiles can then be derived from this distribution to evenly split the questions into 4 levels. An example of this is shown in figure 3.2. There was no way to estimate a lecture's level of difficulty, and so lectures were assigned a default difficulty of '0'. With the level of difficulty incorporated, the action space now extends to a size of 35 (7 parts $\times$ 5 levels of difficulty). Note that lectures have 2 additional part tags (general and unspecified), therefore increasing the action space to 37.

3.3 Data Exploration: Reward function

The reward function ties our agents optimization objective with our domain objective. NLG is commonly used in this field of research [2][7][56] since it is easily measured with tests and provides an succinct quantification of learning gain [10][40]. However, EdNet does not provide any post-test/pre-test record for their users. Therefore, the reward can only be a function of what is present in the logs. With this limitation in



(a) (b)

Figure 3.3: Returns distribution. Reward function: symmetrical (a) punitive (b).

mind, we chose our reward function to be some sort of proxy to student performance and base it on the correctness of a student's response towards an activity. We also integrate the activity's difficulty into the reward design such that correct responses on harder levels indicate stronger performance and attain higher rewards. Inversely, incorrect responses on easier levels attain a larger punishment (negative reward). This is to avoid the agent abusing the reward function by always assigning easy questions. Lectures do not have a correct/incorrect responses and so a default reward of '0' is assigned. One might wonder why an agent will choose an action that always yields a '0' reward. Our argument is that other actions, i.e. presenting a question, have a probability of yielding a negative reward and so in expectation might produce negative rewards. Another argument is that the agent is not myopic and will decide its actions based on the future return (cumulative rewards) rather than the immediate reward.

Initially, a symmetrical reward function was designed with rewards for questions ranging from 1 to 4 if answered correctly or -1 to -4 if answered incorrectly, depending on the level of difficulty. However, this led to a right-skewed cumulative reward distribution across the population of users, suggesting that the reward design was somewhat too generous. As we wanted this distribution to mimic a normal distribution usually exhibited in student grades [24], we doubled the penalty of incorrect answers transforming the range towards -2 to -8. This consequently led to a more bell shaped distribution (see figure 3.3).

3.4 Data Exploration: State features extraction

The dataset in EdNet is longitudinal, where observations are made on the same subjects (users) across a period of time [15]. Most of the state features derived in this section will be longitudinal/temporal in nature too since not much can be derived from a single

isolated observation.

Several studies have shown the significant impact of the representation choice on the final agents performance, with some arguing it is just as influential as optimization algorithm itself [56][51]. This aligns with the theoretical understanding of why human tutors usually outperform their computer counterparts, in that they are able to adapt to certain cues exhibited by the student during learning [61]. The representations consist of features that infer such 'cues' from the dataset. To investigate the impact of the representation on performance, we first create a feature pool, from which different representations are formed using the greedy iterative augmentation algorithm.

We begin with base features that are widely seen in similar RL driven ITS implementations [2][28][56][7]. We also chose these features as they were readily derivable from the coarsest hierarchy, KT1. This sets a reasonable minimum requirement for the data gathering process, should this implementation be repeated with other datasets. Three base features were constructed:

1. **'correct_so_far'** 2. **'av_time'** 3. **'topic_familiarity'**

'correct_so_far' measures the ratio of correct responses to the number of activities attempted. The ratio then needs to be quantized so that a finite MDP can be formed. Four bins were constructed for this purpose, with the bin limits derived from the 1st, 2nd and 3rd quantiles. This is to ensure an equal division of the entire dataset to avoid sparse regions of feature space. 'av_time' is derived based on cumulative average of the elapsed time measured at each activity. This too is quantized into 4 bins.

Although actual topic familiarity is difficult to infer from the logs, a proxy could be derived in the form of the amount of activities covered per topic. However, since there are 7 parts, we would need 7 'sub-features' to measure the current familiarity of each. If each sub-feature has 4 bins, this would quickly lead to an explosion in size of the state space. Therefore, we propose 3 alternative features that could approximate the information captured by 'topic_familiarity' in a more condensed format.

1. **'av_fam'** 2. **'topic_fam'** 3. **'part_fam'**

'av_fam' is the average part familiarity across all the 7 parts. 'topic_fam' only captured the part familiarity of the **previously** chosen action and disregards the familiarity of other parts. 'part_fam', is the dimensionally reduced representation of 'topic_familiarity' using a deep autoencoder. Further details on the autoencoder dimensional reduction implementation is provided in the appendix section 8.1. By evaluating the policy performance of the representations consisting of these three features separately, 'topic_fam' was eventually chosen and included in the base representation.

Feature Pool	Bins	Description
"prev correct"	3	A flag to indicate whether the user answered correctly in the previous question. Fixed value for lectures.
"expl received"	4	Cumulative count of explanations reviewed by the user.
"ssl"	8	A count of the number of steps since the current part was last encountered.
"sec guess"	4	The cumulative count of answer oscillations across the user's history, normalized by the activities consumed.
"lects consumed"	4	A cumulative count of the lectures consumed by a user.
"slow answer"	2	A flag to indicate whether the user's elapsed time for the preceding question was above the average elapsed time for that question.
"steps in part"	4	The cumulative count of how many steps a user has spent in the current part.

Table 3.1: Feature pool (not including 'topic_fam' & 'part_fam')

Following this, 7 more features were constructed and added to a feature pool, table 3.1. These were excluded from the base MDP, since: 1) they were derived from the higher fidelity levels, or 2) they were variants closely related to the base features. These features were inspired by the utility of the domain information it captures. For instance, 'ssl' (or 'steps since last') was designed to proxy a user's 'forgetting' of the associated part, an important factor of learning from a psychological perspective [17]. Like 'part_fam', 'ssl' was dimensionally reduced using an autoencoder, since it was part specific. Meanwhile, 'sec_guess' and 'slow_answer' were proxies to uncertainty/confidence, which are important cues in tutoring [5]. Relative to prior ITS work, our feature space can be considered to be much more larger since existing approaches usually limit the feature size to be binary [2][38][56]. With each user covering on average 440 activities in their learning period [9], a binary split would lose a lot of information on the evolution of a feature value throughout the course. Our features in contrast, have up to 8 bins. This ultimately imposes a necessity for a quality training corpus that can provide sufficient support for each of the many unique combinations within the feature space. This is where the scale of EdNet provides a distinct advantage relative to previous implementations.

Chapter 4

Model Construction and Training

4.1 Model-based Offline RL

4.1.1 Student model MDP construction

With the feature pool derived, the subsequent step in the pipeline is to derive an MDP student model. The methodology chosen is based on the works of [56][52] where the transition probabilities are modelled as multinomial distributions derived from state transition counts observed in the dataset as shown in equation 4.1. Intuitively, this means that a particular outcome s_i, of enacting action a_k in state s_j has a probability given by to the number of times that outcome was observed in the dataset, normalized by the sum of all possible outcomes observed under the same conditions.

$$\hat{p}(s_i|s_j,a_k) = \frac{c(s_i,s_j,a_k)}{\sum_{i=1}^{n} c(s_i,s_j,a_k)} \tag{4.1}$$

Where $c(s_i,s_j,a_k)$ is the count of observed transitions where enacting action a_k in state s_j and leads to the next state s_i. This provides the transition probabilities component of the MDP student model for each (s,a,s') or the three argument dynamic [55]. The other component is the reward function. In many standard MDP definitions [56][52][7], the reward function is also defined in terms of the three argument dynamic i.e. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. This assumes a deterministic environment reward with respect to a given (s,a,s'), which cannot be said about our proposed reward design in section 3.3. Recall that our rewards are a function of the **student's correctness** and the **level of difficulty** and can take values $r \in \{-8,-6,-4,-2,0,1,2,3,4\}$. While the level of difficulty is captured by the action in the (s,a,s') tuple, the correctness is only captured in the states when 'prev_correct' (refer table 3.1) is included in the representation. Without it, the reward function is stochastic with respect to a given (s,a,s') . A

solution for this is to consider the 'four argument' (s,a,r,s') environment dynamics as described by [55] in equation 4.2:

$$p(s',r|s,a) = Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}, \tag{4.2}$$

The estimated dynamics, $\hat{p}(s_i,r|s_j,a_k)$, can be derived from the data, in a similar fashion with the three argument dynamics i.e. equation 4.1, with small changes to the count arguments. Using the four argument dynamics, an expected reward can then be calculated for the three argument reward function, $r(s,a,s') = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s]$ as shown in equation 4.3[55].

$$r(s_i|s_j,a_k) = \sum_{r\in\mathcal{R}} r\frac{\hat{p}(s_i,r|s_j,a_k)}{\hat{p}(s_i|s_j,a_k)} \tag{4.3}$$

Where $\mathcal{R}$ is the discrete set of all the possible rewards (9 in total). With this, our MDP model consisting of the three argument transition probabilities and reward function is defined. Note that we prioritise a three argument dynamic instead of four since it is more standard in relevant literature and is the required format for the Policy Iteration library [12] used in the training the agent.

Another important point to note is that the transition probabilities are entirely dependent on the transition counts observed in the dataset. Consider the case where the transition probability estimate $\hat{p}(s_i|s_j,a_k)$, calculated using equation 4.1, has $\sum_{i=1}^{n} c(s_i,s_j,a_k) = 10$. Compare this to a another estimate but with $\sum_{i=1}^{n} c(s_i,s_j,a_k) = 1000$. It is clear that the latter has much more support (supporting observations) from the dataset and the resulting estimate is more reliable [56]. If the training corpus is small, then such unreliable estimates are hard to avoid. In cases like this, the state feature representation must be simple/coarse enough so that sufficient support can be provided from the small dataset. However, this also means that the pedagogical agent will have limited knowledge (of the student) to respond effectively and might perform poorly in the real world [61]. This is a why a larger training corpus such as EdNet is vital for constructing a sufficiently complex student model, at least under this data-driven MDP methodology. The massive number of observations means that even large and complex representations could have sufficient support to yield reliable transition probability estimates. For the base representation, the support for each unique state in the feature space representation is shown in figure 4.1. Note that the minimum state support within this representation is around 41,000 observations.

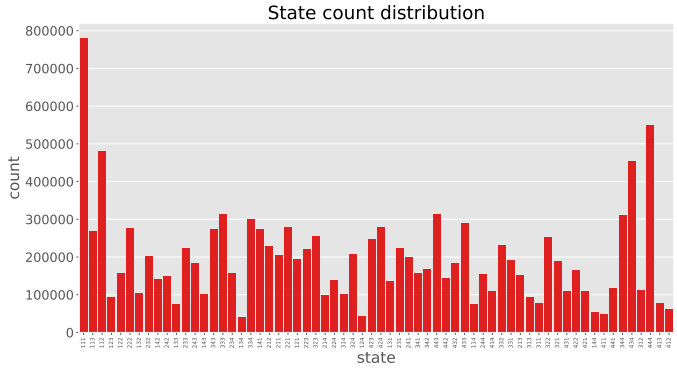


Figure 4.1: Distribution of support for each unique state (integer encoded) in the base MDP

As we continuously augment the representations, the support for each unique state will inevitably fall due to further division of the observations. Another factor in providing a balanced distribution of support within the transitions, is the variety of actions chosen in each state. This ultimately depends on the action space (described in section 3.2) and the behaviour policy used to obtain the dataset. A higher fidelity action space will lead to a blowup in the size transition space i.e. the unique combinations of (s, a, s'). The behaviour policy is the strategy used in taking the actions observed in the dataset. Although it is unknown in most cases, it is important that the behaviour policy is sufficiently varied in terms of its action choices to ensure a balanced distribution of support. Because of this, a random behaviour policy fits the objective well [52]. Since users in EdNet are allowed to select the 'part' and the type of activity they work on [9], we can make an assumption that this random criteria is **partially** fulfilled. The caveat here is that not all users have access to all parts i.e. free users are limited to parts 2 and 5 only.

4.1.2 Training the agent: Policy Iteration

With the data-driven MDP ready, we can utilize a Dynamic Programming (DP) method to train the agent. The DP library by Cross [12] provides two algorithms for this purpose, Policy Iteration (PI) [27] and Value Iteration. Some empirical studies show that the latter posses faster convergence properties [55], however in our applications, PI reached convergence in much fewer steps. Therefore, PI was used in all the DP experiments that follow. PI can be broken down into two repeating phases. The first involves evaluating the value of every state in the finite MDP under the current policy. This is known the Policy Evaluation phase and utilises the Bellman Equation as follows:

$$V(s) = \sum_{s'} p(s'|s, \pi(s))[r(s,a) + \gamma V(s')] \tag{4.4}$$

Where $\pi(s)$ is the action prescribed by the policy π at state s and γ is the discount factor, set at 0.99. Formally, $\gamma \in [0,1)$ is a requirement for continuing MDPs [55] i.e. those without terminating states (more on this in section 5.2). For our purposes, we also do not intend to heavily discount future student performance and therefore set a suitably high γ. The policy at the start is instantiated to a random policy (prescribes equal probability mass to all possible actions). The transition probabilities and reward function are provided by our MDP. The Bellman equation/operator is applied to all states $s \in \mathcal{S}$ repeatedly until the difference between the calculated state value at subsequent iterations dips below a threshold. This convergence property is guaranteed due to the Bellman operator being a contraction mapping with $\gamma \in [0,1)$ [55].

What follows is the second phase of PI, where the current policy is made greedy with respect to the current state values. This, 'policy improvement' phase uses the Bellman Optimality equation to derive a new policy as follows:

$$\pi(s) = argmax_a \sum_{s'} p(s'|s,a)[r(s,a) + \gamma V(s')] \tag{4.5}$$

Once the greedy actions are selected at each state $s \in \mathcal{S}$, a comparison is made between this 'new' policy and the 'old' policy of the previous iteration. If the actions prescribed differ in any states, then another iteration of policy evaluation and policy improvement is performed. The algorithm terminates when the 'new' policy and 'old' policy are identical. At this point, under the policy improvement theorem, the policy is the optimal policy π^* for this MDP [55]. Note that the optimal policy is deterministic.

4.2 Evaluating the policies: ECR

As part of the pipeline, an evaluation metric for π^* is required for each representation at every iteration. Offline policy evaluation (OPE), i.e. the performance estimation of an RL policy without empirical testing, is a field of research in itself, with many proposed techniques and each with their unique advantages/disadvantages [59][46][18][37]. We will pursue a more in-depth analysis of the different applicable OPE techniques in chapter 5. However, as a quick evaluation tool (realtive to other OPEs) for repeated usage in the pipeline, the Expected Cumulative Reward (ECR) metric was chosen.

The ECR computes the average of the expected return (or cumulative reward) under the policy, across all **initial states** in the dataset:

$$ECR = \mathbb{E}_{s_0 \sim \mathcal{D}, \pi^*} Q(s_0, \pi^*(s_0)) \tag{4.6}$$

The $Q(s,a)$ function is simply the 'action value' of taking action a in state s and thereafter following π, and s_0 is the initial state. In each state representation, the initial state of every user in the EdNet is the same. This is because we lack any prior information of the user before they begin the course on *Santa*. Traditionally in ITS implementations, the initial state would capture information from the students pre-test scores and so would vary across the students in the dataset. For EdNet, the ECR is simply the state value of the unique initial state of each representation $ECR = V_{\pi^*}(s_0)$.

4.3 Selecting an Optimum Representation

We now explore a methodology for selecting an 'optimal' representation. As described in the problem statement (section 1.2), the agent's capacity to provide an adaptive learning environment, depends on its ability to leverage information on the student's current cognitive state. This information is inferred from features extracted from the logs. With more features in the representation, one should expect a better approximation of the students cognitive state and consequently a better equipped pedagogical agent to provide effective sequencing. We utilise a greedy 'wrapper' approach in obtaining our optimal representation [51]. This involves a search of the feature space and generation of several candidate feature subsets. Each of these subsets will be evaluated based on its corresponding policy derived from the DP algorithm. The complete procedure is outlined in algorithm 1. Note the limit on the number of features $\mathcal{N}$ can be based off a computational limit or a threshold of minimum support for every unique combination in the feature space.

While our search procedure involves exhaustively looping through every remaining feature in the feature pool Ω to form the subsets, one can alternatively employ a different search algorithm such Monte Carlo tree search [23] or correlation based feature selection [51] to create more informed subsets that are likely to be better candidates. These techniques would be useful in limiting the number of iterations needed in larger feature pools.

Algorithm 1 Greedy iterative feature augmentations

Input: Feature pool Ω, Dataset $\mathcal{D}$, Max. number of features $\mathcal{N}$ (optional)
Set: Optimal feature representation $\mathcal{S}^*$ to be base representation.
while $size(\mathcal{S}^*) \leq \mathcal{N}$ **do**
  for $\omega_i \in \Omega$ **do**
    Set: $\mathcal{S}_i = \mathcal{S}^* + \omega_i$
    $MDP = Construct\_MDP(\mathcal{S}_i, \mathcal{D})$
    $\pi^* = Policy\_Iteration(MDP)$
    $ECR_i = Calculate\_ECR(\pi^*)$
  end for
  Set $\mathcal{S}^* = \mathcal{S}_i$ with highest ECR_i.
  Remove feature from pool $\Omega = \Omega - \omega_i$
end while

4.4 An Alternative: Model-Free Offline RL

In model-based RL, the purpose of the model is to provide an idea of 'what will happen' if we choose actions according to our policy instead of following the observed trajectories. In our case, these estimations are guided by observations in the dataset. However, not all estimations are of equal uncertainty. Some might be more robust than others owing to the stronger support from the dataset. When rolling out our policy from the initial state, our actions might lead to state visitations that are out of distribution (OOD) [62]. These are states that are infrequently observed in the dataset and therefore induce a high level of uncertainty in the model.

If we simply eliminate the model and use **standard** model-free methods directly on the dataset, studies show that the policies will now gravitate towards OOD actions [62][21][33] (further discussed in section 6.2). To combat this issue, several solutions have been proposed, including constraining the target policy to limit its deviation from the behaviour policy (the policy used in gathering the data). This is known as batch constrained RL [22]. However, as our behaviour policy could potentially be random, we are less inclined to constrain our target policy towards it. Instead we utilize another family of solutions which involve regularising the action values of OOD state-action pairs. This solution can be implemented with model-based techniques such as the MOPO algorithm [62] or with model-free algorithms as the Conservative Q-learning (CQL) algorithm [33]. To work within time constraints, we opted for the simpler

model-free CQL. CQL is built on top on the widely popular model-free online Q-learning algorithm [60]. In Q-learning, the agent learns the action value function, $Q(s,a)$, as it interacts with the environment by minimizing the loss $\delta(s,a)$ shown in equation 4.7. The policy is then the argmax of the Q-functions over the actions in every state $s \in \mathcal{S}$.

$$\delta = \|Q(s,a) - (r(s,a) + \gamma \max_{a'} Q(s',a'))\|^2 \tag{4.7}$$

Applying Q-learning directly on the collected dataset will lead to the OOD action issues described above. Therefore, several modifications are introduced in CQL to enable it to perform reliably in the offline setting. Note that in the offline setting, the state action tuples are not obtained by interacting with the environment, but are sampled from the pre-collected dataset $\mathcal{D}$.

$$\delta_{QL} = \mathbb{E}_{s,a\sim\mathcal{D}}[\|Q(s,a) - (r(s,a) + \gamma \max_{a'} Q(s',a'))\|^2] \tag{4.8}$$

$$\delta_{CQL} = \delta_{QL}(s,a) + \alpha(\mathbb{E}_{s\sim\mathcal{D},a\sim\pi}[Q(s,a)] - \mathbb{E}_{s,a\sim\mathcal{D}}[Q(s,a)]) \tag{4.9}$$

CQL adds a regularization term alongside the Q-learning loss, δ_{QL} (see equation 4.9). The minimization of this modified loss expression means that we are actively trying to minimize the erroneously high Q-values in our policy and maximize Q-values of state-action pairs that occur more frequently in the dataset [33]. The end result is a more conservative Q-value estimate which implicitly encourages the policy to stick with more familiar actions (as opposed to the explicit constraints of Batch Constrained RL). Additionally, using CQL also means there is no need to estimate the unknown EdNet behaviour policy, further avoiding a potential source of error [43].

In our CQL implementation, we utilize a publicly available offline RL library, *D3RLPY* [50]. We limit our state features to mirror the optimum representation selected from the pipeline. This is to allow for a fair comparison of the two approaches (model-based vs model-free) during the OPE phase. The CQL function in the library is built on top of a 'DQN' implementation, a deep-learning variant of the Q-learning algorithm [57]. Several configurations were varied in the experiments, including the α hyperparameter in equation 4.9, the presence of batch normalization [29] and dropout [54] in the network architecture and different Q-functions ranging from traditional to distribution reinforcement learning [4][14]. The details of the Double-DQN algorithm and the configuration related parameters such as Dropout, Batch Normalization and Distributional RL can be referred to in their original papers. However, a discussion on their impact on the CQL performance will be conducted in section 6.2.

Chapter 5

Offline Policy Evaluation

Previous RL applications have been successful in domains such as board games [53], video games [41] and robotics [32]. In these fields, the optimal policy (or set of policies) can be evaluated accurately and inexpensively through simulators or by direct interaction with the environment. However, when the domain involves human interaction, the evaluation becomes more challenging, with live interaction being expensive [37]. Developing simulators is also not straightforward as it could potentially induce some form of bias and have questionable real world accuracy, especially when it comes to complex subjects such as the human mind. The problem becomes amplified when there are multiple policies to evaluate (as there usually are), further increasing the cost of empirical evaluations [30]. Therefore, the offline policy evaluation (OPE) field has been developed specifically to address this issue, where an evaluator needs to provide reliable estimates of policy performance using only the collected data [37].

5.1 Importance Sampling

Importance sampling (IS) is a range of methods that in general estimate the expected values under one distributions given samples from another [55]. A wide range of RL literature have adopted this method as a way of evaluating a *target policy* (the policies derived from the RL algorithms) given samples derived from the *behaviour policy* (the policy used to gather the data) [30]. In this work we explore three (IS) variants; Ordinary IS (OIS), Weighted IS (WIS) and Per-Decision IS (PDIS).

$$OIS = \frac{1}{N}\sum_{i=1}^{N}[(\prod_{t=0}^{T_i}\frac{\pi_e(a_t^i|s_t^i)}{\pi_b(a_t^i|s_t^i)})(\sum_{t=1}^{T_i}\gamma^t r_t^i)] \tag{5.1}$$

$$WIS = \frac{\sum_{i=1}^{N}[(\prod_{t=0}^{T_i}\frac{\pi_e(a_t^i|s_t^i)}{\pi_b(a_t^i|s_t^i)})(\sum_{t=1}^{T_i}\gamma^t r_t^i)]}{\sum_{i=1}^{N}\prod_{t=0}^{T_i}\frac{\pi_e(a_t^i|s_t^i)}{\pi_b(a_t^i|s_t^i)}} \tag{5.2}$$

$$PDIS = \frac{1}{N}\sum_{i=1}^{N}[\sum_{t=1}^{T}\gamma^t\prod_{j=0}^{t}\frac{\pi_e(a_j^i|s_j^i)}{\pi_b(a_j^i|s_j^i)})r_t^i] \tag{5.3}$$

Note that N represents the number of users in the dataset and T_i is the trajectory length observed for user i. A key feature in all three variants is the **importance sampling ratio** $\prod_{t=0}^{T_i}\frac{\pi_e(a_t^i|s_t^i)}{\pi_b(a_t^i|s_t^i)}$, which considers the differences in action probabilities between the target policy π_e and the behaviour policy π_b. The product of the individual ratios across $t_i = 0 \rightarrow T$ quantifies whether a given sequence is more (or less) likely under π_e than π_b and therefore weights the returns accordingly. Averaging this across the entire dataset has the effect of adjusting the **expected return** sampled from the distribution generated by π_b to estimate the **expected return** sampled from π_e.

A well documented problem with IS estimators is the high variance induced by the importance sampling ratio due to: 1) a large difference between the two policies or 2) a long horizon length for the trajectories [59]. WIS reduces this variance by utilizing a weighted average instead of the simple average in OIS [55]. PDIS modifies the importance sampling ratio so that the likelihood of a reward at time step t should only depend on the preceding steps (Notice the product in equation 5.3 spans only until the current t). This too has an effect of reducing the variance [55]. All these methods rely on the available knowledge of the behaviour policy. Since we do not have explicit information on this, π_b must be estimated from the dataset, $\mathcal{D}$ as shown in equation 5.4.

$$\hat{\pi}_b(a|s) = \frac{\sum_{s,a\in\mathcal{D}}\mathbf{1}[s=s,a=a]}{\sum_{s\in\mathcal{D}}\mathbf{1}[s=s]} \tag{5.4}$$

5.2 Rollouts: Monte Carlo Policy Evaluation

Alternative to Importance Sampling, 'Direct Methods' focus on regression based techniques to directly estimate the value function under a given target policy [59]. Most of these methods do not need an estimation of the behaviour policy. In this section we implement a model-based direct method, Monte Carlo (MC) Policy Evaluation. This involves performing 'rollouts' from the initial states using the target policy until episode termination. The observed returns from each state in the rollout are averaged across many rollouts to yield the value function of the state. To 'rollout' our policy

we would need to interact with the environment. However, as the name 'model-based' suggests, a model (in our case is the data-derived MDPs) acting as simulator allows us to perform these rollouts offline.

A key requirement for MC policy evaluation is an **episodic environment**, one where the episode terminates at a finite step at a 'terminating state' [55]. Although the user episodes in EdNet are finite, our data-derived MDPs are continuing i.e. without a terminating state. While a default terminating state could have easily been created for this purpose, it is unclear as to which 'action' would transition the final observed state to this terminating state and what 'reward' it would receive in the process. The choice of reward, could inadvertently impact our agents decisions at the earlier states. Hence our MDPs were designed to be continuing to avoid this ambiguity.

This poses a problem with MC policy evaluation since the returns are only calculated when the episode ends. A potential workaround was to manually terminate the episode at a fixed length of rollout and calculate the returns from there. We chose a rollout length of 1000 steps and show that because of the discounting, any reward, $r \in \mathcal{R}$, received past this step, will have a negligible influence on the return of the **initial state** i.e. $max_r(\gamma^{1000}||r \in \mathcal{R}||) \approx 3.5 \times 10^{-4}$. Hence this rollout length provides a good approximation of the long term return, since any future actions will have minimal influence on the value of the initial state, i.e. the only state value of concern in our analysis. However, this assumption will only work with the 'first-visit' variant of MC policy evaluation (equation 5.5), where only the returns of a state when it was first encountered in the episode are considered and averaged across the rollouts [55]. This is opposed to the 'every-visit' variant which considers all the returns from a state every time it is visited in the episode. A fixed rollout length will not be suitable in the latter variant, since the initial state could be encountered more than once during the rollout. For example, if s_0 was encountered again at the 500th time step, then its return estimate for the second visit is based only on the remaining 500 future steps. Implementing the first-visit variant ensures that all $V(s_0)$ estimates are derived from observations spanning 1000 time steps ahead.

$$V_{\pi_e}^{MC}(s_0) = \frac{1}{N_{rollouts}} \sum_{i}^{N_{rollouts}} \sum_{t=1}^{1000} \gamma^t r_i^t \tag{5.5}$$

5.3 Fitted Q Evaluation

The evaluator in the previous section was dependent on a model and therefore can be biased. Furthermore, the model in our case in unable to predict the outcomes of unseen

state-action pairs (in the dataset). This means that if there are any such pairs in the policy then the model-based evaluation might **fail**. For instance consider a step in the rollout where in the current state s, an action a_x is prescribed by the target policy. When the model is required to predict the outcome of enacting action a_x in state s, it will **fail** if (s, a_x) was never observed in the dataset. If there are many unseen state-action pairs in the policy, a model-free OPE, like IS, is preferred. In this section, we explore another model-free estimator, Fitted Q Evaluation (FQE), as a low-variance alternative to IS [59]. FQE fits a Q-function under a given policy based on observations in the dataset [34]. It treats the evaluation as a supervised learning problem and utilises function approximators in the process. In its original implementation FQE learns a sequence of approximators $\hat{Q}(s,a) = \lim_{k\to\infty} \hat{Q}_k(s,a)$, where at each k:

$$\hat{Q}_k = \arg\min_f (f(s_i, a_i) - y_i)^2 \tag{5.6}$$

The target, y_i is defined as $y_i = r_i + \gamma Q_{k-1}(s'_i + \pi_e(s'_i))$ where f is the function approximator. Since, FQE uses function approximation, it can estimate the value of unseen state-action pairs. Also note that the Q-values are approximated only by the observed reward and value of the following state. This method of estimation is known as bootstrapping, and is different to the IS estimators, which utilise the entire return of an episode [55]. By bootstrapping, FQE avoids the compounded noise on the value estimates over long episodes, thereby exhibiting a lower variance [55] than the IS estimators.

We again employ the *D3RLPY* library for our FQE implementation. Its function approximator is a 2-layer neural net with 256 hidden units and ReLU activations. Two separate neural nets are initialized. A 'value-net' is trained at every iteration when the loss is minimized as in equation 5.6. A 'target-net' is trained at specified intervals by performing a hard-copy of the 'value-net' parameters. Note that the target value, y_i is derived using the target-net, not the value-net. This dual network design in deep Reinforcement Learning was popularized by the double Q-learning architecture in Van Hasselt et al. [57], and has since been widely used to avoid the non-stationarity problem [57]. The FQE loss under this function approximator design is shown in expression 5.7, with the value net parameters noted as θ and target net parameters θ'.

$$\delta(\theta) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}}[(Q_\theta(s,a) - (r + \gamma Q_{\theta'}(s', \pi_e(s'))))^2] \tag{5.7}$$

After training the network, we can then use the function approximators to predict the value of the initial state under a given target policy. This value can then be compared across different target policies.

Chapter 6

Results & Discussion

The presentation and discussion of results is broken down into 4 subsections. Section 6.1 will cover results pertaining only to the Policy Iteration algorithm while section 6.2 will cover only CQL related results. Following this, section 6.3 will compare applicable OPE results across both methods and finally section 6.4 will present some notable domain related findings.

6.1 Policy Iteration Results

6.1.1 Expected Cumulative Rewards Across Representations

A summary of the results from the greedy iterative augmentations is provided in table 6.1. The feature description for the corresponding MDP representations are given in table 6.2. Note that the base MDP as discussed in section 4.1.1 is denoted as 'MDP_lp'. The 'ECR Diff.' column shows the percent improvement in ECR relative to

Round	Representation	ECR	ECR Diff. (%)	Representation Size
Base	MDP_lp	238.44	-	64
1	MDP_aug4	283.63	18.95	256
2	MDP_aug46	387.42	36.6	2048
3	MDP_aug468	392.05	1.19	4096
4	MDP_aug4687	396.00	1.01	16384
5	MDP_aug46873	396.00	0	65536

Table 6.1: ECR results showing best performing representation at each iteration

Representation	**Features**
MDP_lp	topic_fam, correct_so_far,av_time
MDP_aug4	topic_fam, correct_so_far,av_time,expl_received
MDP_aug46	topic_fam, correct_so_far,av_time,expl_received, ssl
MDP_aug468	topic_fam, correct_so_far,av_time,expl_received, ssl, prev_correct
MDP_aug4687	topic_fam, correct_so_far,av_time,expl_received, ssl, prev_correct,av_fam
MDP_aug46873	topic_fam, correct_so_far,av_time,expl_received, ssl, prev_correct,av_fam, time_in_part

Table 6.2: Representation Description

the smaller representation preceding it. The 'Representation Size' column illustrates the size of the state feature space. This is dependent on each constituent feature's bin size, i.e. the number of discrete bins allocated. Note that results presented here are only showing the best performing representation at each round of the feature augmentation. The full results of the greedy iterative augmentations, including the preliminary experiment to decide the base MDP, are provided in the appendix (table 8.2).

The largest spike in ECR followed at the second round of augmentations with the addition of the 'ssl' feature with an increase of 36.6% over the preceding representation. 'ssl' measures the number of steps or activities (questions/lectures) consumed since the current part was last encountered. This feature is inferring the 'forgetting' element during the learning process and was inspired by the 'spacing effect' described in Ebbinghaus [20]. Early research in instructional sequencing in language learning used models of forgetting to great success [1]. Our findings concur with this, in that by including 'ssl' into the feature space, we dramatically increased the agent's performance. One could argue that this ECR increase was more influenced by the larger bin allocation to 'ssl' (8 relative to 4 for most other features) rather than the actual utility of the domain information it is measuring. However, if that were the case, then we would expect 'ssl' to be the first feature added to the base representation. This was not the case since the best performing feature in the first round of augmentations was 'expl_received', a 4-size bin feature. Nonetheless, further exploration is needed to concretely rule out the factor of the larger bin size.

At the final round of iteration, the best performing representation 'MDP_aug46873'

only equals the performance of preceding representation 'MDP_aug4687'. Though we did not have a specified limit imposed on the number of features, $\mathcal{N}$, the performance plateau exhibited at this final round indicated a suitable termination point for the augmentation algorithm. And since 'MDP_aug4687' produced equal performance to 'MDP_aug46873' with a smaller representation size, it was chosen to be the optimum representation within this feature pool. Notice that 'av_fam' (a rejected candidate for topic familiarity in the baseline representation) was reintroduced into the feature pool before the greedy iterations algorithm commenced and was automatically selected at the fourth round of augmentations. This was done to leverage its interpretable properties for further domain related analysis such as in section 6.1.5 (unlike 'part_fam' which is less interpretable due to the autoencoded feature space).

6.1.2 Policy Support Analysis

The optimal policy derived from the Policy Iteration algorithm is deterministic, where a single action is prescribed at each state. Meanwhile, the estimated behaviour policy (see section 5.1) is stochastic, where a distribution across actions is provided at each state. A comparison of the two policies, one from 'MDP_lp' (denoted as PI) and the behaviour policy (denoted as BP) is shown in table 6.3. Note for comparison purposes, the most common action (with greatest probability mass) in each state in displayed from the stochastic BP policy. 10 random states were sample for this comparison. The actions under each policy is integer encoded and can be interpreted from the action encoding dictionary in the appendix (table 8.3). The most common actions under the BP policy are mostly within part 5. This is because a large portion of the user population (free users) are limited to certain parts. 'PI Prob' illustrates the conditional probability of observing the prescribed action under 'PI Policy' given the state in the dataset. 'PI Support' provides the actual number of observations in the dataset supporting the $(s, \pi_{PI}(s))$ tuple in the dataset. 'State Support' is the number of times the state was observed in the dataset. Note that $PI\,Prob = \frac{PI\,Support}{State\,Support}$. 'BP Prob' is the probability mass of the most common action under the stochastic behaviour policy.

From some states, we can observe very low supporting observations for actions prescribed by the PI Policy. This finding demonstrate that model-based methods can be very statistically biased. In a noisy environment like human learning, the policy's performance might be vulnerable to the uncertainties of the model. An investigation into the sensitivity of the different representations to noise is performed in section 6.1.5.

State	PI Policy	PI Prob	State Support	PI Support	BP Policy	BP Prob
212	35	0.000079	304915	24	22	0.172337
344	4	0.028779	384241	11058	24	0.11656
141	32	0.000486	372520	181	24	0.378449
211	35	0.000086	266083	23	22	0.159525
223	35	0.000082	291366	24	24	0.146513
123	9	0.045657	133190	6081	24	0.22811
132	1	0.008887	143803	1278	24	0.262401
444	4	0.025812	675238	17429	29	0.0617
312	35	0.000076	144888	11	22	0.160441
234	3	0.011005	211817	2331	24	0.130688

Table 6.3: Comparison of policy support for 'MDP_lp' (PI) and behaviour policy (BP) from 10 randomly sampled states.

6.1.3 Penalising Uncertainties: Avoiding Unseen Actions in the Policies

Continuing this analyses, we proceed to check if any policies were enforcing unseen state-actions pairs, with 0 support from dataset. This should not occur under policies derived from tabular methods as explained in section 2.1. By default, our PI algorithm prescribes state-action pairs a value of 0 if it was never observed. Upon inspection, we discovered that some of the larger representations yielded policies with unseen actions. The policy derived from 'MDP_aug4687' prescribed unseen actions in 10 states. While this was a small fraction of the total state space (around 65,000), unseen actions are an important issue to address because, in the tabular case, any state-action value estimates must be derived only from related experiences [55].

We discovered that the problematic states had very little support in the dataset and were only observed transitioning to themselves, before the episode ends. In the few times the state was visited, a negative or 0 reward was produced. Since these states would only transition to themselves, the values of these valid actions were either negative (or 0). Hence, from the algorithms perspective, an invalid (unseen) action with a default value of 0, was preferable (or equal) to the observed actions.

To combat this issue, we modified the MDP representations to strongly penalise the rewards from unseen state-action pairs, in the form of a -9999 reward. This dis-

couraged the agent from choosing such actions even if the only valid actions yielded 0 or negative returns (The worse case is bounded from below at $\sum_{k\to\infty}\gamma^k \times (-8)$ and is clearly higher than $\sum_{k\to\infty}\gamma^k \times (-9999)$). With these changes in place, we observe no unseen actions in any of the policies. The performance rank of representations remained constant with the ECR changes almost negligible (see table 8.2 in the appendix). This is because the states involved were observed very infrequently and occupied a small probability mass in the transition probabilities. Although this fix was not significant to our results, we wanted to demonstrate a technique in handling the uncertainty induced by unseen or out of distribution (OOD) actions in model-based RL. Interestingly, research by [36] implemented a 'pessimistic policy iteration' that similarly penalises insufficiently supported state-action tuples (filtered by a threshold). Note the analyses that follow will utilise the penalised representations.

6.1.4 Monte Carlo Policy Evaluation

In this section we evaluate the policies under the MC Policy Evaluation. A curve is plotted for the returns (cumulative rewards) from the initial state as the rollout progresses until the 1000th step for a total of 100 rollouts. The 95% confidence intervals are plotted around the mean value of the rollouts. This analysis is shown in figure 6.1. As with the ECR, we can see that the improvements start to diminish significantly after the second round of augmentations. This graph also adds another dimension to the analysis by showing how the value of s_0 changes as the episode progresses. The 95% interval windows also demonstrate how the policy's performance might vary owing to the stochasticity of the environment as exhibited in the dataset. The performance of the estimated stochastic **behaviour policy** under this simulation is also illustrated as a baseline. The values of in figure 6.1 shows the expected student performance from the initial state under a given policy of instructional sequencing. From this comparison, one can expect much better student performance under the RL policies than the baseline, signalling that the adaptive behaviour of the agent under RL framework is superior than the strategies used in the behaviour policy. We can also conclude that the larger representations exhibit better performance, potentially owing to a better approximation of the cognitive state as hypothesized.

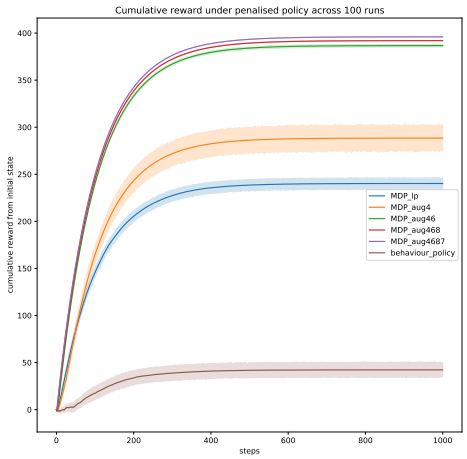


Figure 6.1: Returns from the initial state s_0 as the episode progresses under the policies from the associated representations

Feature to Perturb	Strong	Weak
Topic_fam	$\omega_{s'} > \omega_s$	$\omega_{s'} = \omega_s$
Correct_so_far	$\omega_{s'} \geq \omega_s$	$\omega_{s'} < \omega_s$
Avg_time	$\omega_{s'} \leq \omega_s$	$\omega_{s'} > \omega_s$

Table 6.4: Domain perturbation filters,ψ for each feature in $\bar{\Omega}$ for the 'Strong' and 'Weak' perturbed MDPs, $\bar{\mathcal{P}}$ respectively

6.1.5 Evaluating Performance with Different Student Types

A follow up is to test the robustness of the original policies under perturbations of the environment dynamics. These perturbations are domain informed and are designed to correspond to 'stronger' and 'weaker' students types. Algorithm 2 was created to introduce these domain informed perturbations.

The filter ψ is a set of domain informed filters for each perturbed feature. In our implementation we perturb three base features that were common in all representations i.e. "topic_fam", "correct_so_far" and "av_time". The domain rules for the two separate perturbations 'Strong' and 'Weak' are defined in table 6.4. For example, take the feature "correct_so_far" and the 'Strong' user case. Here we set the filter to capture transitions where the next state s' registers a greater or equal value relative to the current state s. When this filter is inputted in algorithm 2, the transitions that satisfy this filter will be boosted by the constant c. This ultimately has the effect of increas-

Algorithm 2 Domain informed perturbations

Input: Set of features to perturb $\bar{\Omega}$, MDP transition probabilities $\mathcal{P}_{MDP}$, set of domain filters for each feature ψ, positive perturbation constant $c = 0.05$

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

$$\Delta_{s,a,s'} = p_{s,a,s'} + \sum_{\omega \in \bar{\Omega}} \Delta_{\omega} \text{ Where } \Delta_{\omega} = \begin{cases} c, & \text{if } \omega_s, \omega_{s'} \text{ satisfies } \psi_{\omega} \\ 0, & \text{else} \end{cases}$$

end for

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

$\Delta_{s,a,s} = \Delta_{s,a,s} - \frac{1}{|\psi|} \sum_{s'} \Delta_{s,a,s} \; \forall \Delta_{s,a,s'}$

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

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

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

end for

$p_{s,a,s'} = \frac{p_{s,a,s'}}{\sum_{s'} p_{s,a,s'}} \; \forall p_{s,a,s'}$

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

ing the probability mass of this transition, perturbing the original MDP to make such transitions more likely. The results of performing these two separate perturbations are visualised by performing the MC policy evaluation algorithm with the original policy but under the perturbed MDPs (Strong & Weak) as the simulators.

Figure 6.2 shows the results of this analysis for the different representations. Notice that in all the representations, the original MDP always yielded the best performance. This is expected, since the original policy was derived to perform optimally on the original MDP. However, as the representation size increases, the effects of the perturbations becomes less pronounced, almost becoming negligible past 'MDP_aug46'. To determine if the larger representation would be affected with more features perturbed, we conducted another round of perturbations, this time only on 'MDP_aug4687' and with **all** of its features (barring 'ssl') perturbed. The new domain filters are shown in the appendix (table 8.5). Surprisingly, the performance of the policy as shown in the bottom right panel in figure 6.2 is not affected by the perturbations. This could mean that the larger representations are more robust towards deviations from the expected dynamics derived from the data. Hence, we can have more confidence that such policies would be robust in the real-world setting, maintaining its performance for students that exhibit different learning characteristics from our observations in EdNet.

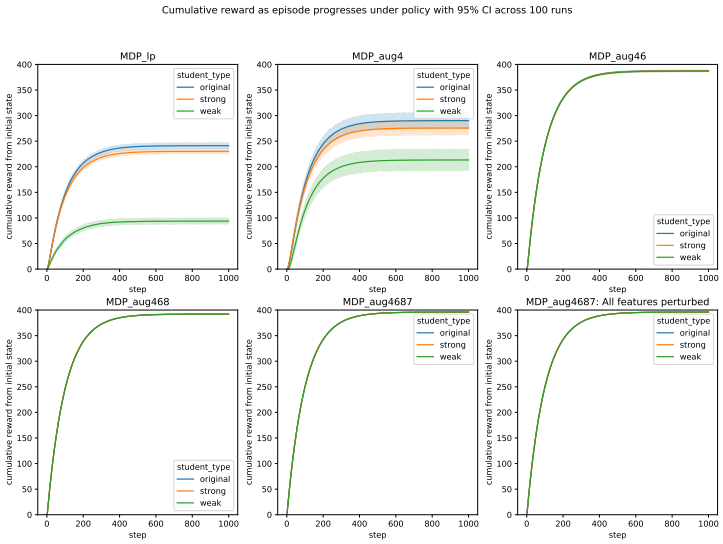


Figure 6.2: MC Policy Evaluation of the original policy under the perturbed MDPs

6.2 Conservative Q-learning

Following our investigation into the model-based policies, we shift our attention to the model-free CQL results. Several runs of the CQL algorithm were performed **using the features from the 'MDP_aug4687' representation as input features**. The dataset was split to a 80:20 train-validation ratio and the ECR and TD-error metric on the validation set is measured during training as shown in figure 6.3. The run are encoded and their hyperparameter settings are given in the appendix (table 8.4). The TD-error measures the level of overfitting observed in the function approximators [50].

$$error_{TD} = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}}[(Q_\theta(s,a) - (r + \gamma \max_{a'} Q_\theta(s',a'))^2] \tag{6.1}$$

We first compare the influence of the Q-function on the results. The mean Q-function (CQL_0) registered a higher ECR however, reported similarly high TD-errors. The latter fact would indicate that the ECR estimation is potentially unreliable. The other two Q-functions, Quantile Regression (QR) (CQL_1) and Implicit Quantile Networks (IQN) (CQL_2) are from the family of distributional Reinforcement learning [14][13]. Briefly, these methods predict a distribution of returns rather than a single expected return as we have seen so far. The reason for doing so is improved stability and convergence when used with function approximations [4]. Indeed when examining the TD-error we can observe that the QR and IQN runs are plateauing in TD-error near the 10th epoch while CQL_0's error continues to rise. We selected the QR model

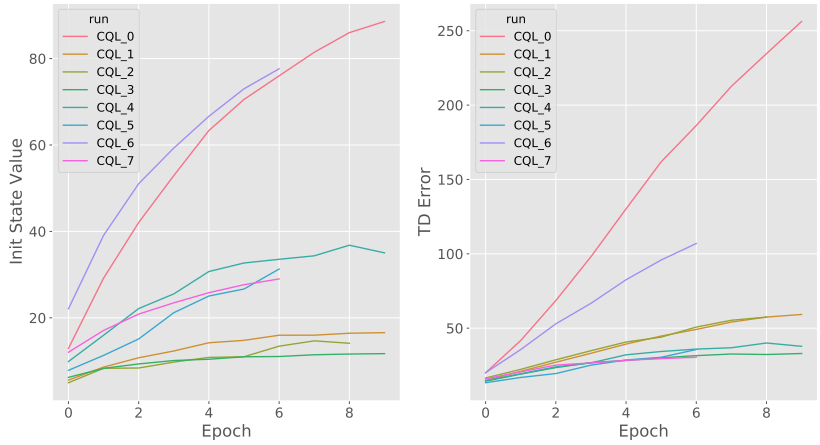


Figure 6.3: CQL Results. Left: Initial State Value Estimate (ECR). Right: TD-Error. Note some runs were terminated before completion due to the server run time limit.

due to its favourable balance between ECR and TD-error and proceed with testing the impacts of dropout (rate 0.5) and batch normalization. Adding these features to the network architecture both improved the ECR and lowered the TD-error.

Next, we wanted to observe the influence of the α hyperparameter in the CQL algorithm (see equation 4.9). Lowering α to 0.5 (CQL_5) yields marginal difference in the results, however setting it at 0 (CQL_6) shows a dramatic increase in both ECR and TD-error. In CQL_6, we are effectively running a standard DQN algorithm on the dataset, removing the conservative element in CQL. Conversely, increasing α to 5 (CQL_7) reduces the TD-error and ECR i.e. making the model more conservative. Surprisingly, we discovered that implementing dropout has a strong impact the variety of actions prescribed, with CQL_4 opting to choose only 2 of the 37 actions in the action space. Further analysis showed that some actions were highly dominant in the EdNet (see figure 8.3), due to the parts limitation imposed on free users. This action imbalance and the regularization effect of dropout led to a limited action variety in the policies. Meanwhile policies without dropout were fairly varied in action choices.

Finally, we wanted to see the extent of OOD actions within the policy space. The results were extreme relative to the PI policies, with the runs registering thousands of OOD actions in their policies, the worst being the standard DQN variant with 5500 (see full results in the appendix in table 8.6). In retrospect, this is not surprising since value function approximators combined with offline RL is heavily prone to OOD actions [21]. Indeed the purpose of function approximation is to generalize across unseen

states/state-action pairs. The problem as pointed out by Levine et al. [35], is that the values of these unseen state-actions are usually overestimated in the offline setting. In online RL, these overestimations would be corrected once the agent begins to query these OOD actions. Algorithms such as CQL were developed to minimize these offline overestimations and reduce the attractiveness of such actions. This claim is observed to be true in our results, where increasing α has an inverse effect on the number of OOD actions. To this extent, we can conclude that the CQL algorithm has fulfilled its purpose. However, the natural question is whether such conservative behaviour is beneficial within our domain of ITS. OOD actions are not necessarily bad if the domain does not impose strict restrictions on action choices. Nonetheless they are challenging to accurately evaluate, requiring empirical testing and policy deployment with actual users. However, we can get some insights into the **relative** policy performance using the data we have. This leads us to the results in Offline Policy Evaluation.

6.3 Offline Policy Evaluation

Three OPE methods were presented in section 5, MC Policy Evaluation, Importance Sampling and Fitted Q Evaluations. ECR is also technically an OPE method, and from this metric alone, the PI policies significantly outperform the CQL policies and the behaviour policy (where the ECR is simply the average of the returns in the dataset at 7.4). However, the big caveat is that ECR is statistically inconsistent for the model-based methods owing to the bias induced by the student model [37]. MC Policy Evaluation was already discussed for the PI policies in section 6.1.4. Unfortunately, the same methodology cannot be used to evaluate the CQL policies as the large number of OOD actions exhibited in the policy prevents us from doing so, as explained in section 5.3. Therefore, to fairly compare the two approaches, we implement Importance Sampling and Fitted Q Evaluations.

6.3.1 Importance Sampling

The results of several policies from each approach are shown in table 6.5. For reference, the behaviour policy achieves an average return of 7.4. Immediately, we can see that OIS and PDIS have wildly varying results. An inspection into the variance of the associated IS estimators reveals extreme figures especially within the model-based policies. This variance does decrease as the representation increases in size, indicat-

	OIS	OIS Var	PDIS	PDIS Var	WIS
MDP_lp	-2.21E+05	2.43E+15	3.86E+09	5.92E+23	-4.146
MDP_aug4	-1.03E+06	5.35E+16	6.43E+16	2.07E+38	-0.940
MDP_aug46	-6.96E+04	2.42E+14	8.50E+03	3.58E+12	-4.382
MDP_aug468	6.39E-02	2.21E+02	-7.67E+00	3.53E+06	4.319
MDP_aug4687	6.52E-02	1.88E+02	3.97E-01	2.49E+03	4.910
CQL_0	6.00E+00	9.92E+04	4.24E+05	9.02E+15	3.871
CQL_1	6.28E+00	1.04E+05	-2.93E+00	1.31E+06	4.055
CQL_4	1.09E+01	1.03E+06	-1.04E+04	8.53E+12	6.772
CQL_6	3.86E-03	1.32E-01	1.48E+02	2.36E+09	0.003
CQL_7	1.09E+01	1.03E+06	-1.03E+04	8.53E+12	6.751

Table 6.5: Importance sampling results for model-based & model-free policies

ing a more level performance across the set of users, concurring with the analysis in section 6.1.5. The WIS exhibits less noisy results owing to its weighted averaging procedure. Here we can observe that the larger MDP representations for the most part, show better performance, The CQL policies barring CQL_6 (the standard DQN) in general perform better than the model-based policies, possibly indicating an advantage of the conservative behaviour. However, these results must be taken with a grain of salt, since it is well known that IS estimators perform poorly for long horizon episodes [59] like that observed in EdNet with T_i (in equation 5.1) averaging 440 steps. Together with the observed strong disparity between π_b and π_e i.e. ($\frac{\pi_e}{\pi_b} << 1$) [1] and the product of the IS ratios, $\prod_{t=0}^{T_i} \frac{\pi_e(a_t^i|s_t^i)}{\pi_b(a_t^i|s_t^i)}$, vanishes to near zero.

6.3.2 Fitted Q Evaluation

Considering the unstable IS results, we turn to the FQE estimator as suggested by Voloshin et al. [59]. FQE was shown to be a more appropriate offline evaluation tool for long horizon episodes with larger policy mismatches [59]. The estimates of the initial state value under the respective policies, $Q(s_0, \pi(s_0))$, are given for several policies in figure 6.4. The losses (equation 5.6) for each run are also provided. The performance of the estimated behaviour policy (BP) is included as a reference baseline. The PI

[1] Note that a large policy mismatch might also yield ($\frac{\pi_e}{\pi_b} >> 1$). However, this is less likely to happen, since π_b would have to be very low i.e. making such observations rare in the dataset.

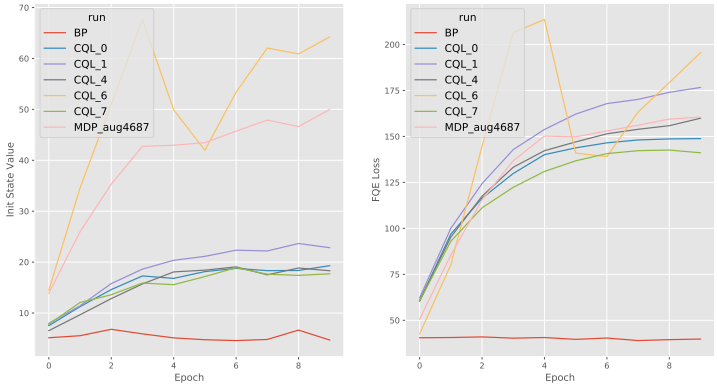


Figure 6.4: Left: FQE results on initial state values, $Q(s_0, \pi(s_0))$. Right: FQE loss

policy derived from 'MDP_aug4687' is superior to most other CQL policies. CQL_6 (the standard DQN) produced the highest $Q(s_0, \pi(s_0))$ estimate of all policies but its validation loss was relatively higher (and more unstable) than the PI and CQL policies, thereby introducing more uncertainty in its $Q(s_0, \pi(s_0))$ estimate. The BP policy was significantly outperformed by the RL policies, although the same caveat with respect to the losses apply here too, since the BP loss was far lower than the rest. In general, the FQE results show that the RL policies are superior to the baseline BP. However, more tuning on the FQE architecture/hyperparameters is necessary to reduce the losses, thereby ensuring a level ground comparison.

6.4 Domain Related Findings

By observing the state values and policies derived from the RL algorithms, we can discover interesting insights in how the agent perceives the information in the states and how it behaves accordingly. In figure 6.5, we plot the derived state values against two features in 'MDP_lp'. Based on our reward design, the state values indicates the future user performance. From the agent's perspective, the expected future performance is much higher when the student has a high correct-incorrect answer ratio. However, the relationship between the average time is more complex. At higher values of 'correct_so_far', a higher 'av_time' entails a larger state value, but when 'correct_so_far' is low, the opposite is true. Even if the policies themselves are not used, findings like this can inform us of useful features and their relationships in predicting future user performance.

Looking in the action choices in the policies, we discover that the RL algorithms

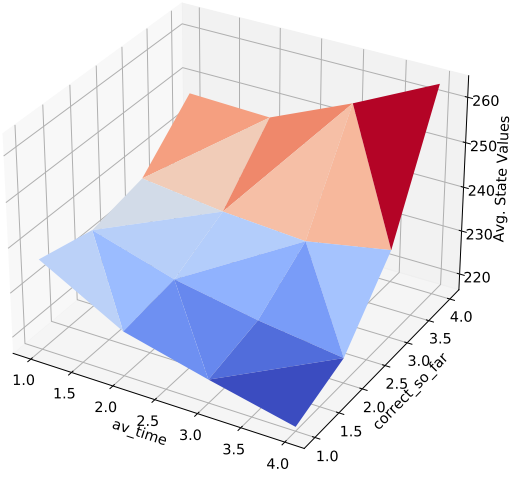


Figure 6.5: State values vs features

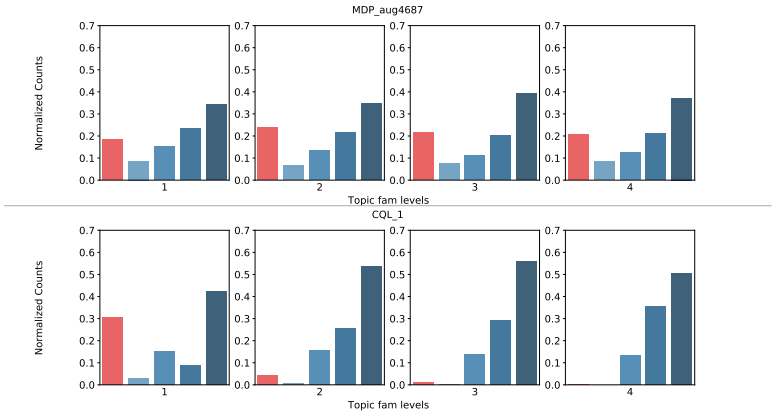


Figure 6.6: Conditional action distribution given the feature value for topic_fam. The x-axis captures the feature value of topic_fam with 1 being the lowest and 4, highest. Red bars represents lecture, while blue represent questions at increasing levels of difficulty.

tend to put preference on question level 4 actions. Indeed these do yield the highest reward and the lowest punishment in our reward design. One possible extension is to investigate how a change in the reward function design would impact the policy preferences. A conditional distribution on the action selection for a given feature and its values is shown in figure 6.6. The top figure shows the analysis for the CQL_1 policy while the bottom for the PI policy of MDP_aug4687. For both policies, we can observe that at all levels of 'topic_fam', the policy predominantly chooses the hardest question. Interestingly, for CQL_1 the policy decreasingly prescribes lectures when the topic familiarity goes up. Again even if the policies are not deployed, such patterns can be useful as a technique in letting the data guide pedagogical strategies. The results for other features and other policies are given in the appendix section 8.3.

Chapter 7

Conclusions

In this project we approached the challenge of designing an adaptive RL based pedagogical agent that can provide an optimized sequencing of learning materials to maximize learning. Training an RL agent with actual users is far too resource intensive. Therefore, we simultaneously tackle the problem of training and evaluating an RL algorithm offline based only on pre-collected data. A purely data-driven student model was created for this purpose. We hypothesized that a complex model is required to capture the intricacies of human learning. To investigate this theory, a large dataset, EdNet, was necessary to provide sufficient support for the models.

Our student model was constructed in the form of a data-derived MDP, with the transition and reward dynamics estimated from the observations in the data. The raw logs were transformed into domain inspired features. An action set was established to delineate our agent's possible controls and a reward function was designed to incentivise the agent in maximizing learning. Without NLG, we base our measurement of learning on the perceived student performance. By using the MDPs we then trained our agents with the model-based Policy Iteration algorithm. To determine whether a more complex model yields better tutoring, we employed a greedy iterative augmentation procedure. The ECR metric guided how we chose our features and demonstrated the positive relationship between representation complexity and policy performance. In our analyses we discovered issues with Out of Distribution actions in the policies and presented a solution in the form of penalising rewards. We further evaluated our policies using a modified Monte Carlo Policy Evaluation algorithm and tested their robustness against domain informed perturbations of the dynamics. We show that the larger representation are less impacted by the perturbations and therefore can provide a more equal learning experience for stronger or weaker students.

We then explore a model-free offline RL alternative. The issue of Out of Distribution actions in offline training were discussed and the Conservative Q-learning algorithm was eventually chosen. Several CQL runs were performed under different hyperparameter settings and Q-functions. To make fair comparisons between the model-free and model-based methods, we utilise two Offline Policy Evaluations techniques, Importance Sampling and Fitted Q Evaluations. We demonstrate that at least one of the two RL methods outperformed the baseline in all of the policy performance metrics[1].

Throughout the course of this project, several limitations were acknowledged which consequently opened up to further investigations. The influence of the bin-size on feature preference in the representations was discussed briefly but lacked conclusive evidence to rule out entirely. This work is necessary to ensure that the features are selected based only on the utility of the domain information it captures. From our model-based policy analyses we also discovered OOD actions in the policy space. Though we managed to remedy the problems for completely unseen actions through strong penalisation, the next course of action is to also penalise low supported actions/states **variably** according to their uncertainty as was explored by [36][62]. Finally, our FQE estimator was limited by the varying losses between the different algorithms. While this prevented a definitive ranking between the algorithms, a longer training time with no hardware limitations, plus more work on tuning the parameters/architecture should yield equal losses, consequently addressing this problem.

[1]Barring the unstable Importance Sampling metric, where the baseline behaviour policy achieved a marginally higher value.

Bibliography

[1] R. C. Atkinson. Optimizing the learning of a second-language vocabulary. *Journal of experimental psychology*, 96(1):124, 1972.

[2] J. Bassen, B. Balaji, M. Schaarschmidt, C. Thille, J. Painter, D. Zimmaro, A. Games, E. Fast, and J. C. Mitchell. Reinforcement learning for the adaptive scheduling of educational activities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2020.

[3] J. Beck, B. P. Woolf, and C. R. Beal. Advisor: A machine learning architecture for intelligent tutor construction. *AAAI/IAAI*, 2000(552-557):1–2, 2000.

[4] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

[5] K. Bhatt, M. Evens, and S. Argamon. Hedged responses and expressions of affect in human/human and human/computer tutorial interactions. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 26, 2004.

[6] J. Capper. E-learning growth and promise for the developing world. *TechKnowLogia*, 2(2):7–10, 2001.

[7] M. Chi, K. VanLehn, and D. Litman. Do micro-level tutorial decisions matter: Applying reinforcement learning to induce pedagogical tutorial tactics. In *International conference on intelligent tutoring systems*, pages 224–234. Springer, 2010.

[8] M. Chi, K. VanLehn, D. Litman, and P. Jordan. Inducing effective pedagogical strategies using learning context features. In *International conference on user modeling, adaptation, and personalization*, pages 147–158. Springer, 2010.

[9] Y. Choi, Y. Lee, D. Shin, J. Cho, S. Park, S. Lee, J. Baek, C. Bae, B. Kim, and J. Heo. Ednet: A large-scale hierarchical dataset in education. In *International Conference on Artificial Intelligence in Education*, pages 69–73. Springer, 2020.

[10] V. P. Coletta and J. J. Steinert. Why normalized gain should continue to be used in analyzing preinstruction and postinstruction scores on concept inventories. *Physical Review Physics Education Research*, 16(1):010108, 2020.

[11] A. T. Corbett, K. R. Koedinger, and J. R. Anderson. Intelligent tutoring systems. In *Handbook of human-computer interaction*, pages 849–874. Elsevier, 1997.

[12] M. J. Cross. Markov decision processes (mdp) toolbox, Jan 2015.

[13] W. Dabney, G. Ostrovski, D. Silver, and R. Munos. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, pages 1096–1105. PMLR, 2018.

[14] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos. Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[15] P. Diggle, P. J. Diggle, P. Heagerty, K.-Y. Liang, P. J. Heagerty, S. Zeger, et al. *Analysis of longitudinal data*. Oxford university press, 2002.

[16] F. A. Dorça, L. V. Lima, M. A. Fernandes, and C. R. Lopes. Comparing strategies for modeling students learning styles through reinforcement learning in adaptive and intelligent educational systems: An experimental analysis. *Expert Systems with Applications*, 40(6):2092–2101, 2013.

[17] S. Doroudi, V. Aleven, and E. Brunskill. Where's the reward? *International Journal of Artificial Intelligence in Education*, 29(4):568–620, 2019.

[18] M. Dudík, J. Langford, and L. Li. Doubly robust policy evaluation and learning. *arXiv preprint arXiv:1103.4601*, 2011.

[19] E. Duffin. E-learning and digital education-statistics & facts. *Retrieved December*, 22:2019, 2019.

[20] H. Ebbinghaus. *Über das gedächtnis: untersuchungen zur experimentellen psychologie*. Duncker & Humblot, 1885.

[21] S. Fujimoto, D. Meger, and D. Precup. Where off-policy deep reinforcement learning fails. 2018.

[22] S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

[23] R. Gaudel and M. Sebag. Feature selection as a one-player game. In *International Conference on Machine Learning*, pages 359–366, 2010.

[24] T. Grosges and D. Barchiesi. European credit transfer and accumulation system: An alternative way to calculate the ects grades. *Higher Education in Europe*, 32 (2-3):213–227, 2007.

[25] R. K. Hambleton, R. J. Shavelson, N. M. Webb, H. Swaminathan, and H. J. Rogers. *Fundamentals of item response theory*, volume 2. Sage, 1991.

[26] W. Holmes, M. Bialik, and C. Fadel. Artificial intelligence in education. *Boston: Center for Curriculum Redesign*, 2019.

[27] R. A. Howard. Dynamic programming and markov processes. 1960.

[28] A. Iglesias, P. Martínez, R. Aler, and F. Fernández. Reinforcement learning of pedagogical policies in adaptive and intelligent educational systems. *Knowledge-Based Systems*, 22(4):266–270, 2009.

[29] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[30] S. Ju, S. Shen, H. Azizsoltani, T. Barnes, and M. Chi. Importance sampling to identify empirically valid policies and their critical decisions. In *EDM (Workshops)*, pages 69–78, 2019.

[31] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[32] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[33] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

[34] H. Le, C. Voloshin, and Y. Yue. Batch policy learning under constraints. In *International Conference on Machine Learning*, pages 3703–3712. PMLR, 2019.

[35] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[36] Y. Liu, A. Swaminathan, A. Agarwal, and E. Brunskill. Provably good batch off-policy reinforcement learning without great exploration. *Advances in Neural Information Processing Systems*, 33:1264–1274, 2020.

[37] T. Mandel, Y.-E. Liu, S. Levine, E. Brunskill, and Z. Popovic. Offline policy evaluation across representations with applications to educational games. In *AAMAS*, volume 1077, 2014.

[38] Y. Mao. One minute is enough: Early prediction of student success and event-level difficulty during novice programming tasks. In *In: Proceedings of the 12th International Conference on Educational Data Mining (EDM 2019)*, 2019.

[39] R. Mason and F. Rennie. *Elearning: The key concepts*. Routledge, 2006.

[40] D. E. Meltzer. Normalized learning gain: a key measure of student learning. 2002.

[41] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[42] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[43] A. Nair, M. Dalal, A. Gupta, and S. Levine. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[44] C. Pojen, H. Mingen, and T. Tzuyang. Junyi academy online learning activity dataset: A large-scale public online learning activity dataset

from elementary to senior high school students. *Dataset available from https://www.kaggle.com/junyiacademy/learning-activity-public-dataset-by-junyi-academy*, 2020.

[45] V. Pong, S. Gu, M. Dalal, and S. Levine. Temporal difference models: Model-free deep rl for model-based control. *arXiv preprint arXiv:1802.09081*, 2018.

[46] D. Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80, 2000.

[47] F. E. Ritter, J. Nerb, E. Lehtinen, and T. M. O'Shea. *In order to learn: How the sequence of topics influences learning*. Oxford University Press, 2007.

[48] J. Rowe, B. Pokorny, B. Goldberg, B. Mott, and J. Lester. Toward simulated students for reinforcement learning-driven tutorial planning in gift. In *Proceedings of R. Sottilare (Ed.) 5th Annual GIFT Users Symposium. Orlando, FL*, 2017.

[49] A. Segal, Y. B. David, J. J. Williams, K. Gal, and Y. 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.

[50] T. Seno. d3rlpy: An offline deep reinforcement library. `https://github.com/takuseno/d3rlpy`, 2020.

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

[52] S. Shen and M. Chi. Reinforcement learning: the sooner the better, or the later the better? In *Proceedings of the 2016 conference on user modeling adaptation and personalization*, pages 37–44, 2016.

[53] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[54] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[55] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[56] J. R. Tetreault and D. J. Litman. A reinforcement learning approach to evaluating state representations in spoken dialogue systems. *Speech Communication*, 50 (8-9):683–696, 2008.

[57] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[58] G. Veletsianos and G. S. Russell. Pedagogical agents. In *Handbook of research on educational communications and technology*, pages 759–769. Springer, 2014.

[59] C. Voloshin, H. M. Le, N. Jiang, and Y. Yue. Empirical study of off-policy policy evaluation for reinforcement learning. *arXiv preprint arXiv:1911.06854*, 2019.

[60] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[61] B. P. Woolf. *Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning*. Morgan Kaufmann, 2010.

[62] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Zou, S. Levine, C. Finn, and T. Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.

Chapter 8

Appendix

8.1 Deep Autoencoder Implementation

A deep autoencoder was used to dimensionally reduce the 7-dimensional features 'part_fam' and 'ssl' to 1-dimensional features. The autoencoder's encoding architecture is shown in figure 8.1. The full architecture consist of the encoder plus a decoder. The encoder has two dense layers, a (1 to 2) dense layer and a bottleneck layer. Batch normalization is performed after the first dense layer. ReLU activations are used at each layer. The bottleneck layer performs the compression to 1 dimension. The decoder mirrors this architecture but in the reverse direction, taking the 1-dimensional output from the bottleneck layer and transforming it back into the original 7-dimensions.

The target value of the data is the input value itself. By designing the loss to be difference between the original input value and the reconstructed input value at the output of the decoder, we can train the neural net to perform a compression of the original input feature. The loss measures how well the compression retains useful information, to allow the decoder to reconstruct the original input. An example of the training and validation loss is shown in figure 8.2. After training, this compression functionality is obtained by extracting only the encoder from the complete autoencoder architecture. Our implementation is a simple Keras based autoencoder trained on the 7-dimensional 'part_fam' and 'ssl' separately and limited to data from the first 100,000 users in EdNet. This is to shorten the training time required.

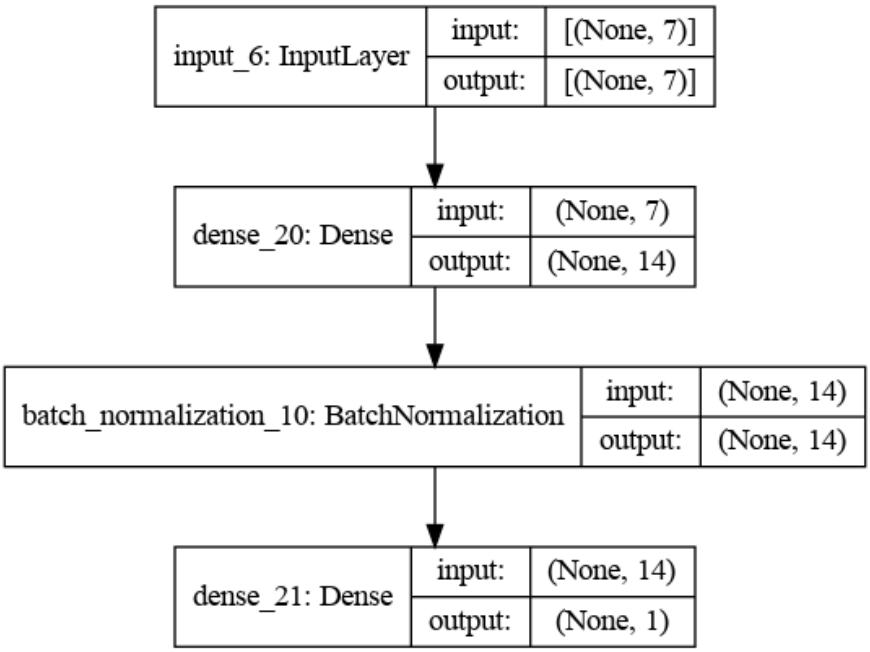


Figure 8.1: Autoencoder architecture (encoder only)

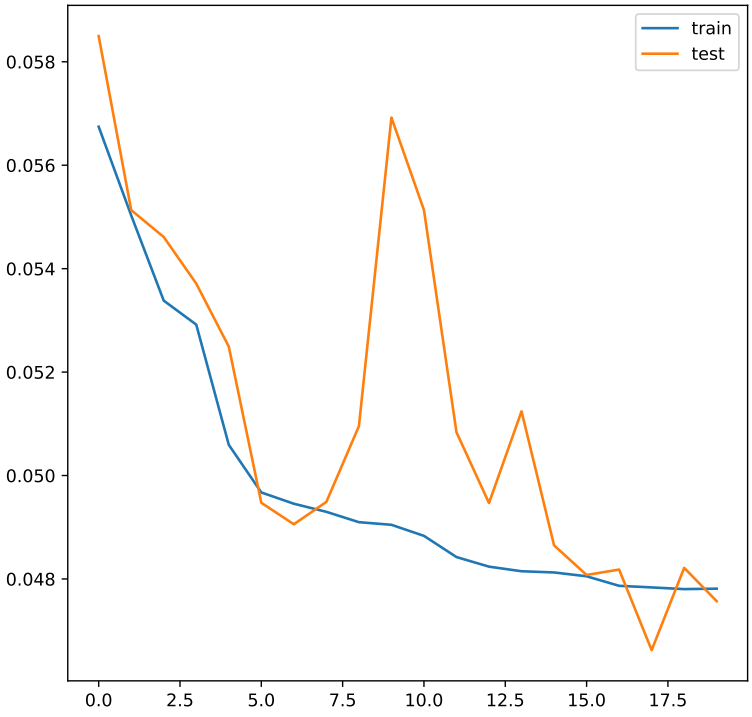


Figure 8.2: Autoencoder (MSE) loss in training for 'part_fam'

8.2 Main Appendix

timestamp	solving_id	question_id	user_answer	elapsed_time
1.57E+12	1	q5012	b	38000
1.57E+12	2	q4706	c	24000
1.57E+12	3	q4366	b	68000
1.57E+12	4	q4829	a	42000
1.57E+12	5	q6528	b	59000
1.57E+12	6	q4793	a	58000
1.57E+12	7	q6488	a	35000
1.57E+12	8	q356	b	23000
1.57E+12	9	q1382	c	22000
1.57E+12	10	q830	b	25000
1.57E+12	11	q11259	b	23000
1.57E+12	12	q1092	a	20000
1.57E+12	13	q524	a	22000
1.57E+12	14	q485	c	21000
1.57E+12	15	q11261	b	23000
1.57E+12	16	q1329	b	20000
1.57E+12	17	q1204	c	18000
1.57E+12	18	q4891	b	28000
1.57E+12	19	q5477	a	35000

Table 8.1: A slice of the log from user 'u1' from KT1

State representation	Start State	ECR	ECR_penalised
MDP_lp	111	238.44	238.44
MDP_av	111	210.75	210.75
MDP_ae	811	213.14	213.14
MDP_aug1	1110	240.14	240.14
MDP_aug2	1111	231.49	231.49
MDP_aug3	1111	255.14	255.14
MDP_aug4	1111	283.63	283.63
MDP_aug5	1111	232.74	232.74
MDP_aug6	1116	257.22	257.22
MDP_aug7	1111	237.90	237.90
MDP_aug8	1112	253.57	253.57
MDP_aug41	11110	340.16	340.16
MDP_aug42	11111	375.10	375.10
MDP_aug43	11111	354.85	354.85
MDP_aug45	11111	337.90	337.90
MDP_aug46	11116	387.42	387.42
MDP_aug47	11111	362.85	362.85
MDP_aug48	11112	368.52	368.52
MDP_aug461	111160	390.19	390.19
MDP_aug462	111161	390.49	390.49
MDP_aug463	111161	391.35	391.35
MDP_aug465	111161	389.84	389.84
MDP_aug467	111161	391.42	391.42
MDP_aug468	111162	392.05	392.05
MDP_aug4681	1111620	393.14	393.14
MDP_aug4682	1111621	394.93	394.93
MDP_aug4683	1111621	395.15	395.15
MDP_aug4685	1111621	393.45	393.45
MDP_aug4687	1111621	396.00	396.00
MDP_aug46871	11116210	394.98	394.98
MDP_aug46872	11116211	395.10	395.10
MDP_aug46873	11116211	396.00	396.00
MDP_aug46875	11116211	394.66	394.66

Table 8.2: Full results from greedy iterative augmentations. ECR_penalised are ECR values from the penalised versions of the same MDPs.

Action Code	**Part**	**Level**	**Action Code**	**Part**	**Level**
0	1	0	**19**	4	4
1	1	1	**20**	5	0
2	1	2	**21**	5	1
3	1	3	**22**	5	2
4	1	4	**23**	5	3
5	2	0	**24**	5	4
6	2	1	**25**	6	0
7	2	2	**26**	6	1
8	2	3	**27**	6	2
9	2	4	**28**	6	3
10	3	0	**29**	6	4
11	3	1	**30**	7	0
12	3	2	**31**	7	1
13	3	3	**32**	7	2
14	3	4	**33**	7	3
15	4	0	**34**	7	4
16	4	1	**35**	0	0
17	4	2	**36**	-1	0
18	4	3			

Table 8.3: Action Encoder. Action level ’0’ indicates lecture. Part ’0’ and ’-1’ are specific for lectures only and denote general and unspecified parts respectively

Run Code	Q-function	Batch Norm	Dropout Rate	α
CQL_0	Mean	FALSE	NA	1
CQL_1	QR	FALSE	NA	1
CQL_2	IQN	FALSE	NA	1
CQL_3	QR	FALSE	0.5	1
CQL_4	QR	TRUE	0.5	1
CQL_5	QR	TRUE	0.5	0.5
CQL_6	QR	TRUE	0.5	0
CQL_7	QR	TRUE	0.5	5

Table 8.4: CQL run code dictionary

Feature to Perturb	Strong	Weak
Topic_fam	$\omega_{s'} > \omega_s$	$\omega_{s'} = \omega_s$
Correct_so_far	$\omega_{s'} \geq \omega_s$	$\omega_{s'} < \omega_s$
Avg_time	$\omega_{s'} \leq \omega_s$	$\omega_{s'} > \omega_s$
expl_received	$\omega_{s'} > \omega_s$	$\omega_{s'} \leq \omega_s$
av_fam	$\omega_{s'} > \omega_s$	$\omega_{s'} = \omega_s$
prev_correct	$\omega_{s'} = 1$	$\omega_{s'} = 0$
av_fam	$\omega_{s'} > \omega_s$	$\omega_{s'} = \omega_s$

Table 8.5: Specific to MDP_aug4687 only. Domain perturbation filters,ψ for each feature in $\bar{\Omega}$ for the 'Strong' and 'Weak' perturbed MDPs, $\bar{\mathcal{P}}$ respectively

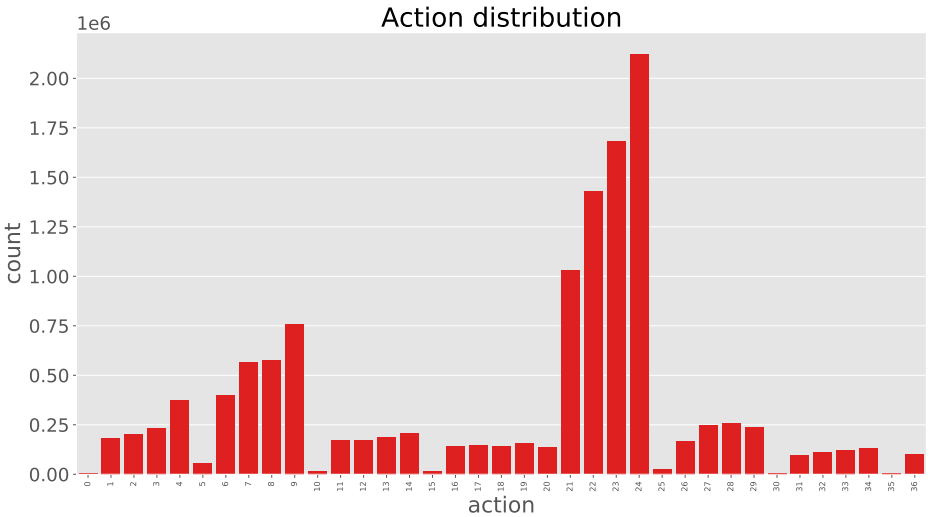


Figure 8.3: Action distribution observed in EdNet

Policy	No. of OOD Actions
MDP_lp	0
MDP_aug4	0
MDP_aug46	1
MDP_aug468	3
MDP_aug4687	10
MDP_lp	0
MDP_aug4	0
MDP_aug46	0
MDP_aug468	0
MDP_aug4687	0
CQL_0	1723
CQL_1	1868
CQL_2	1994
CQL_3	2338
CQL_4	2334
CQL_5	2358
CQL_6	5541
CQL_7	1963

Table 8.6: OOD actions in different policies.

8.3 Conditional Action Distribution under Policies

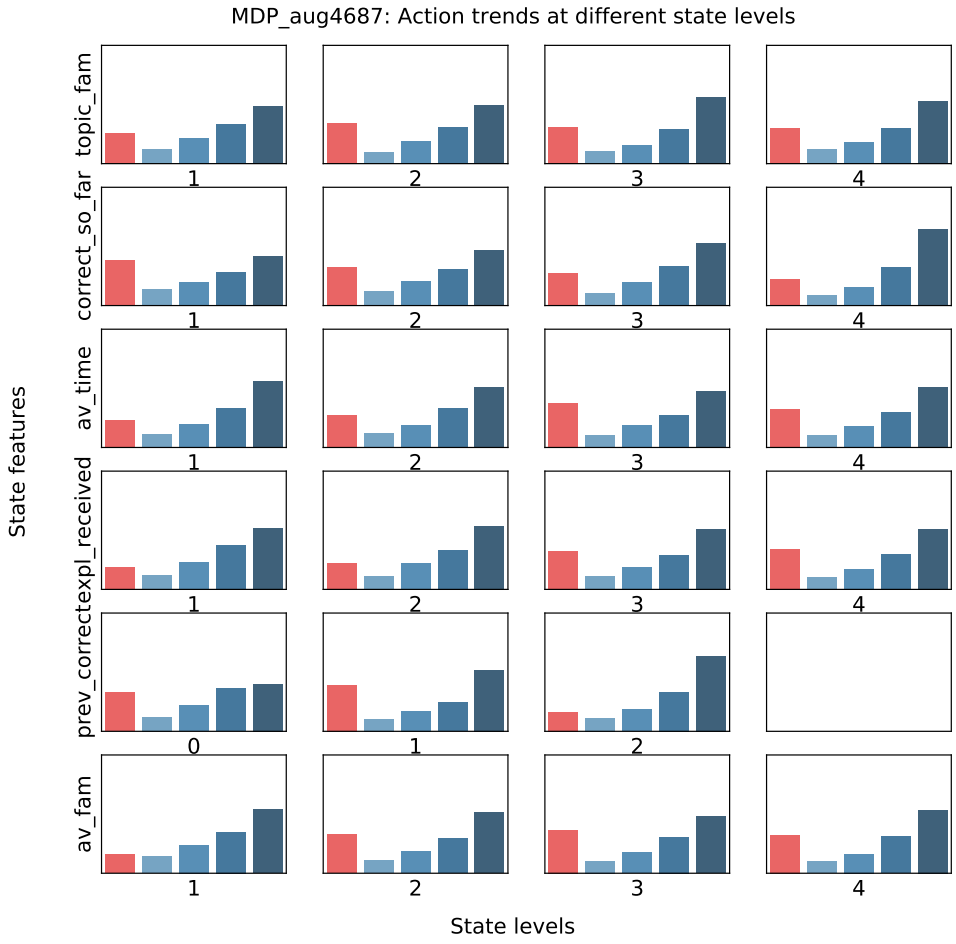


Figure 8.4: Conditional action distribution given a feature value for MDP_aug4687

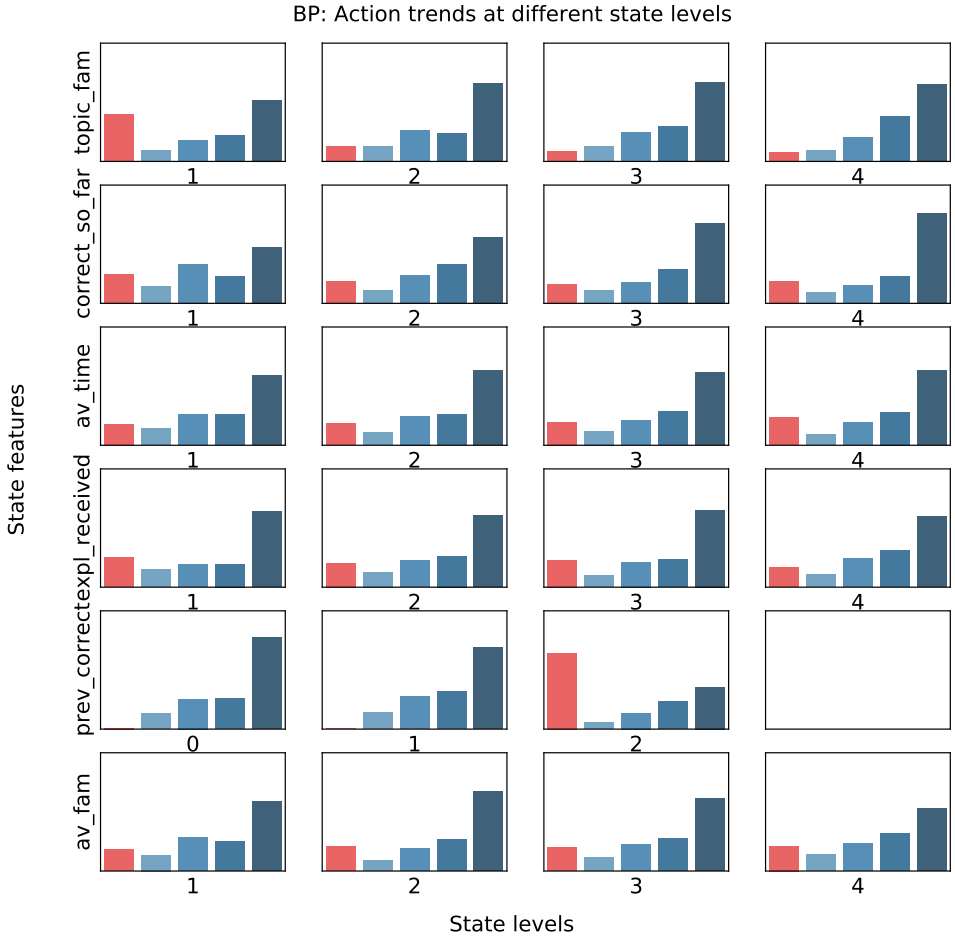


Figure 8.5: Conditional action distribution given a feature value for Discrete Behaviour Policy

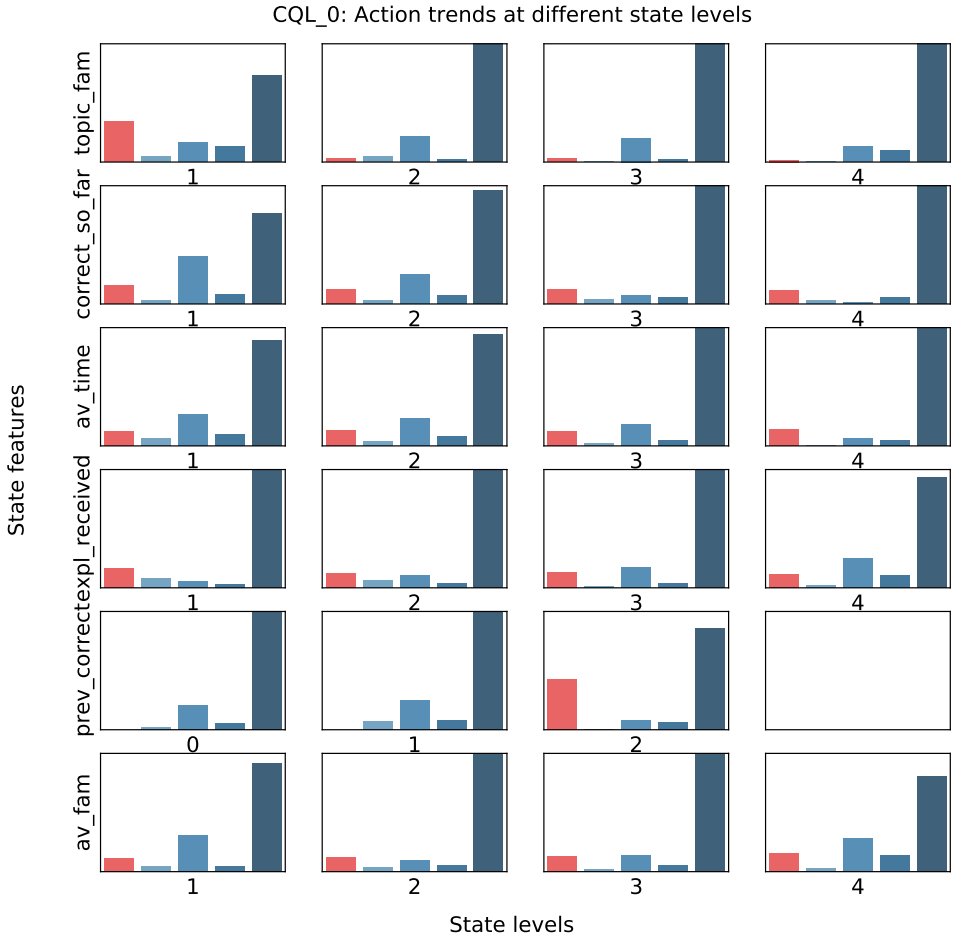


Figure 8.6: Conditional action distribution given a feature value for CQL_0

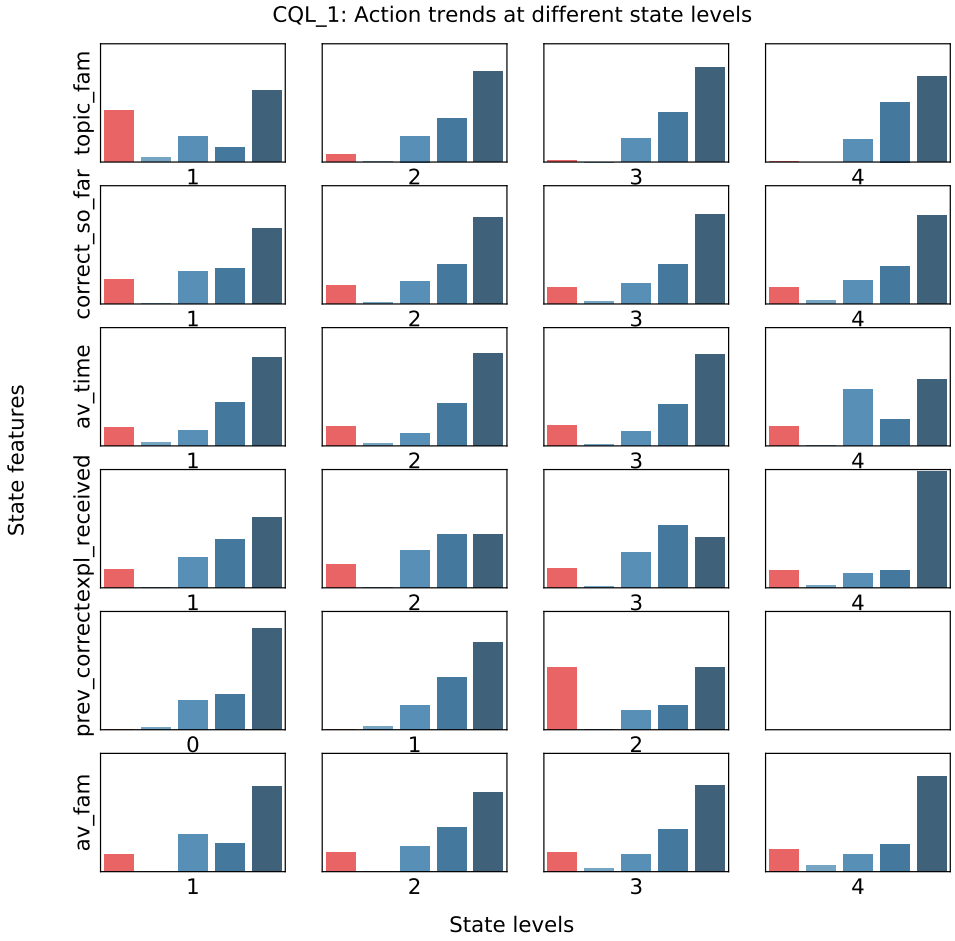


Figure 8.7: Conditional action distribution given a feature value for CQL_1

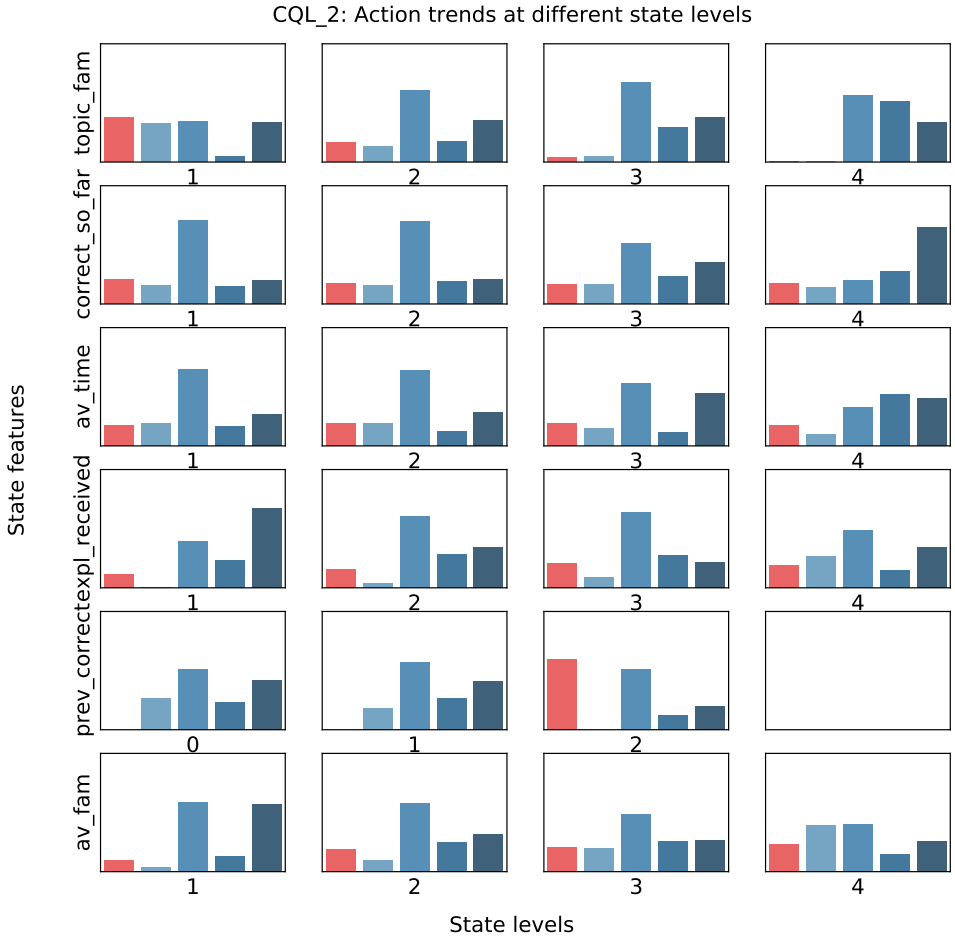


Figure 8.8: Conditional action distribution given a feature value for CQL_2

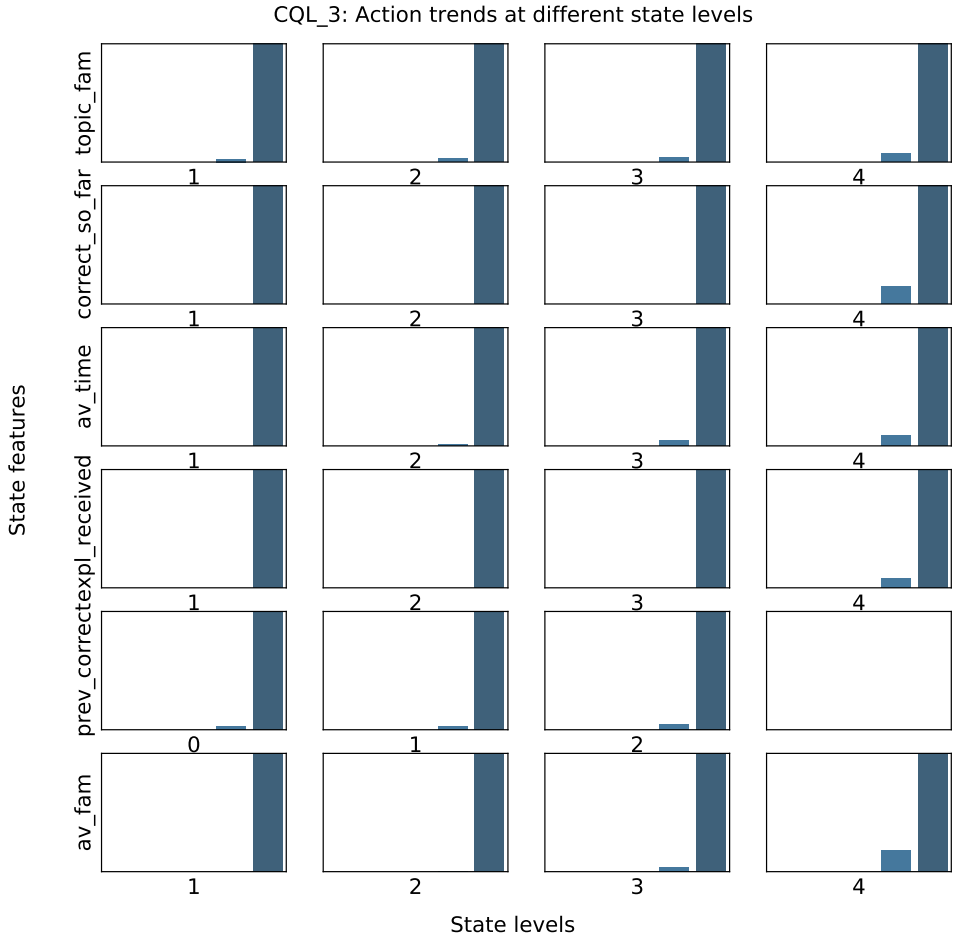


Figure 8.9: Conditional action distribution given a feature value for CQL_3

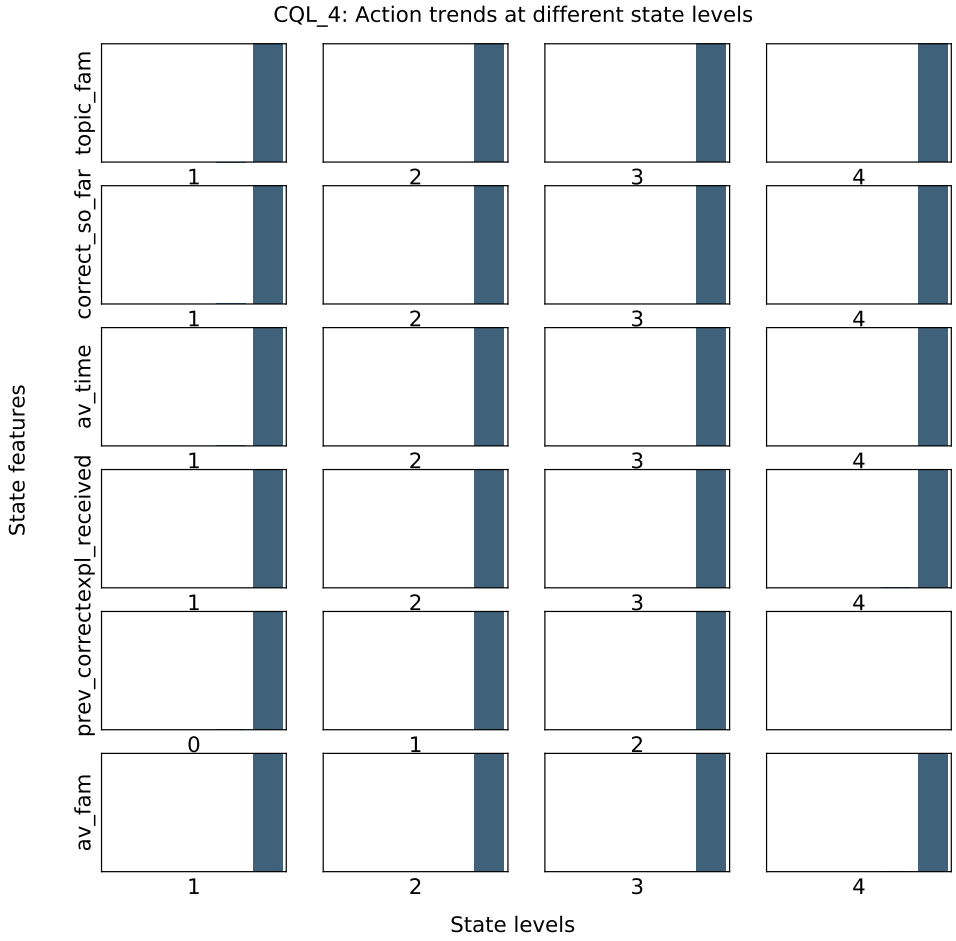


Figure 8.10: Conditional action distribution given a feature value for CQL_4

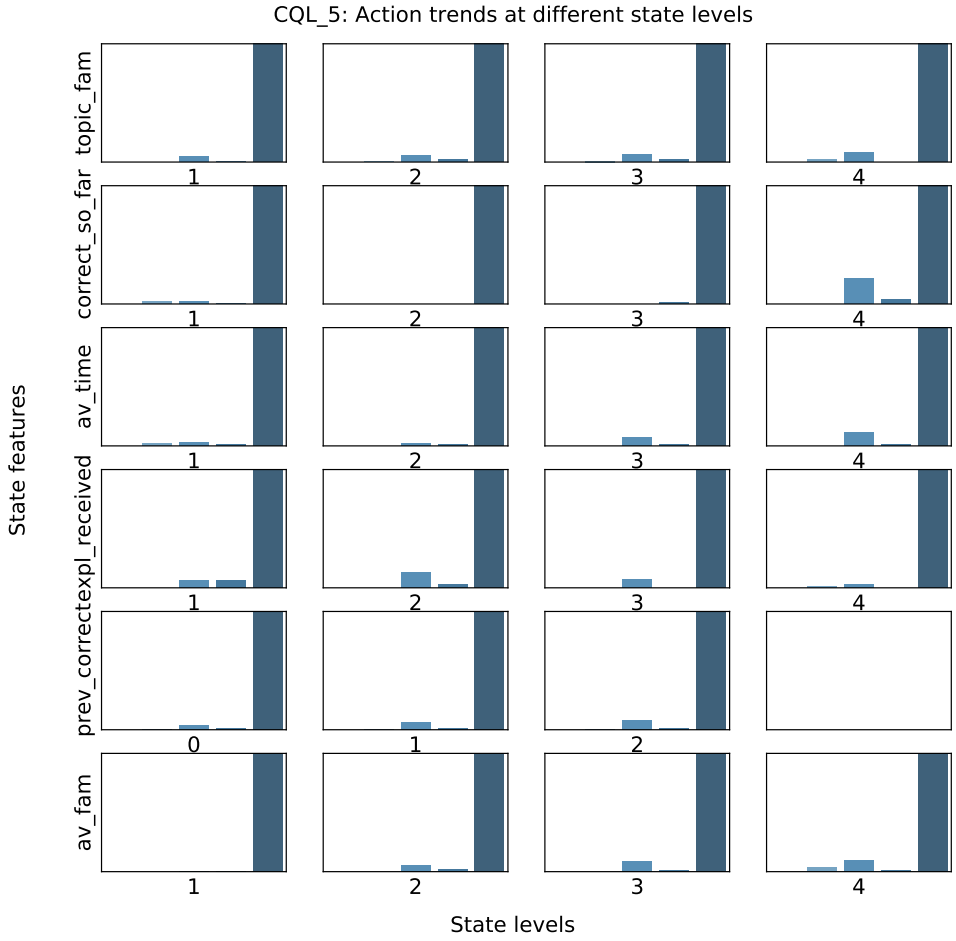


Figure 8.11: Conditional action distribution given a feature value for CQL_5

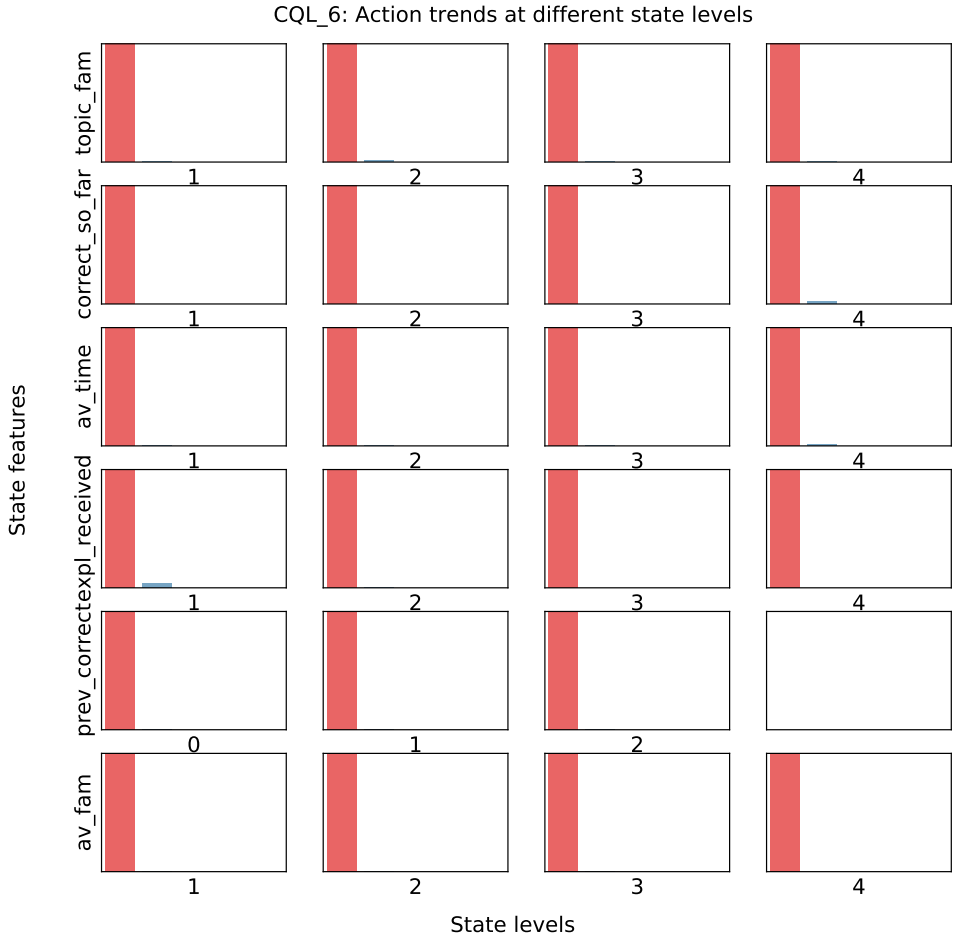


Figure 8.12: Conditional action distribution given a feature value for CQL_6

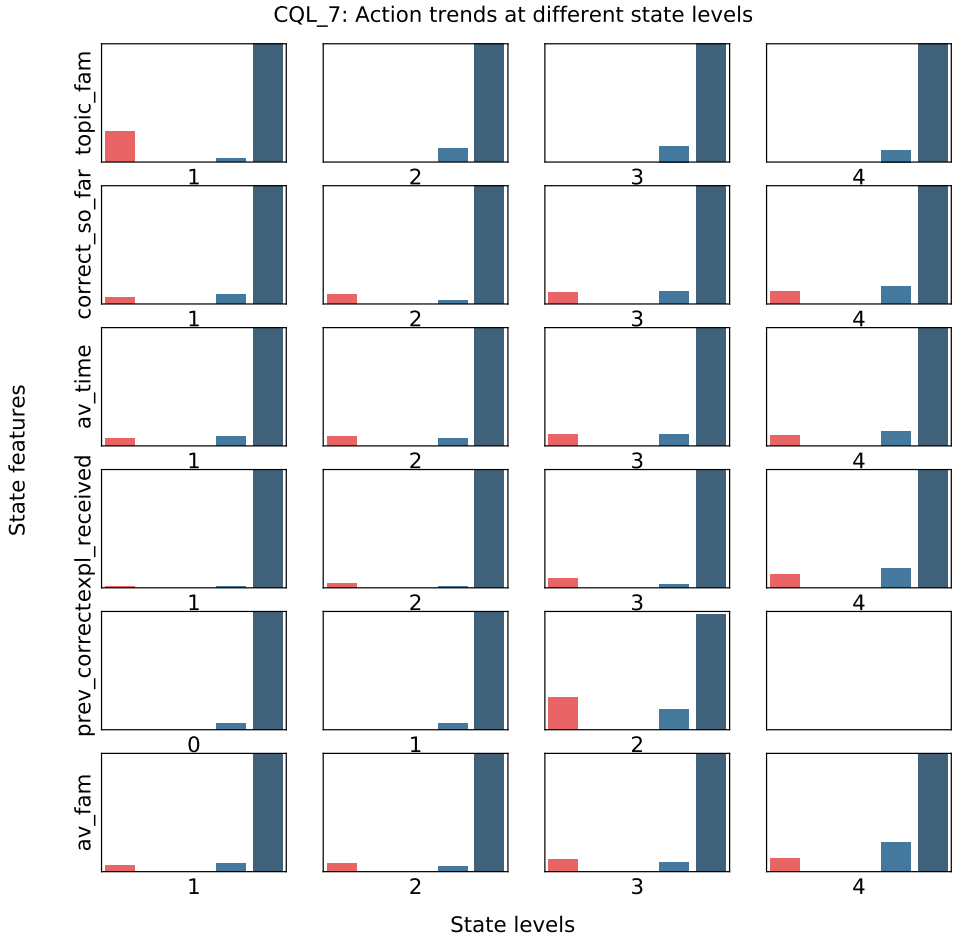


Figure 8.13: Conditional action distribution given a feature value for CQL_7