CS5180 - Project

Exploration on the paper: Learning MDPs from Features: Predict-Then-Optimize for Sequential Decision Making by Reinforcement Learning

Sean Yu

Github link: https://github.com/seanjyu/experimentation-on-decision-focused-RL

Contents

1 Introduction

In this project I aim to explore the paper "Learning MDPs from Features: Predict-Then-Optimize for Sequential Decision Making by Reinforcement Learning" by Wang, et al. This paper was originally presented in NEURIPS 2021.

The link to the code used for this project is here: https://github.com/seanjyu/experimentation-on-decision-focused-RL

1.1 Previous Work

This paper was inspired by a previous paper called 'Smart predict then optimize'. This paper comes from operations research and the original problem that it was addressing was that the typical approach to decision-making involves training a model with data to produce valuable intermediate results, which are subsequently utilized in a separate process aimed at optimizing the final decision.

A simple example of this would be finding the shortest path on google maps. This problem could be formulated as a graph problem with the optimal path being the path with the shortest cost. The naive solution would only consist of the distance of the path but in reality there are many other factors that could affect the 'cost' of a path such as traffic, weather, date, etc. Once the cost for each edge is calculated a separate algorithm can be used to find the shortest path (e.g. Dijkstra's algorithm, linear programming).

A potential problem with the traditional approach is that the training of the model that takes in the input data is independent to the model that creates the decisions. To solve this the 'Smart predict-then-optimize' framework uses the decision errors as part of the loss function in training the model that takes in the data.

1.2 Contributions of this paper

The key contribution of this paper lies in the application of the 'Smart predict then optimize' framework to sequential decision making. This approach involves learning the features of a Markov Decision Process (MDP) to estimate specific unknown parameters (e.g. unkown rewards, unknown transition probabilities) of the MDP and then employing reinforcement learning (RL) to obtain optimal decisions. To link the errors from the RL model to the feature model, an offline off-policy evaluation is used to calculate the errors from the RL model, and these errors are backpropagated all the way to the feature model.

Depending on the environment, backpropogating the errors can be computationally expensive. Therefore, the authors uses a sampling technique to unbiasedly estimate the derivative and also a low rank approximation is used to calculate the derivatives (specifically hessian matrix of the cost function of the found policy).

A concrete example for this framework would be given a grid world scenario in which the goal state is unknown but for each state in the grid there is a set of data that describes the rewards/penalties for each state. To solve this first the data would be fed into a model to predict the rewards/penalties for each state. Then a reinforcement learning algorithm can be used on the environment with the predicted parameters. Once this algorithm is run, a policy is obtained and then off-policy policy evaluation (OPE) can be performed on the policy to obtain the errors of the decision process. Then the errors from this can be back propagated to the to model that was used to predict the rewards of environment.

1.3 Breaking down the framework

1.3.1 Further Explanation

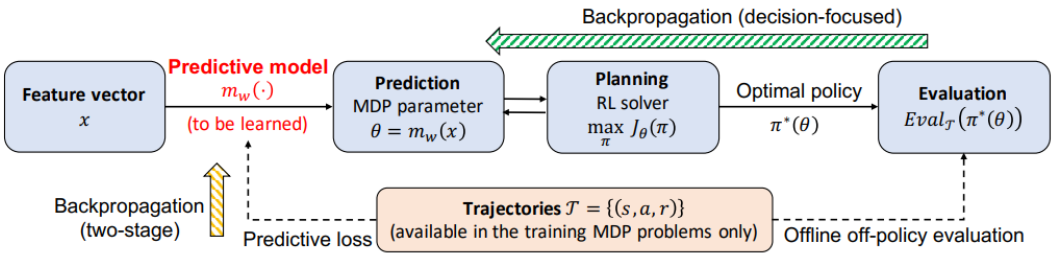


Figure 1: Diagram of the predict-then-optimize framework applied to sequential decision making

The framework was briefly outlined in the previous paragraphs but calculation details will be explained here. Figure 1 shows the steps for the framework. There are two inputs of the framework. Namely, features or data that describe the environment

and episodes (sequence of state, action and rewards) from the environment. The goal of the framework is to maximize the decision quality under an MDP environment with unknown parameters given there exists a set of features or data that can be used to describe these parameters. The maximization is achieved by factoring in the policy's quality when estimating weights for the model predicting unknown MDP parameters.

Regarding the specific steps, first the features are used to predict the unknown parameters for the MDP environment. Therefore the features should ideally have some correlation with the unknown MDP parameter. An example could be given an MDP with unknown transition states each state could have a value assigned to it, these values are fed into a model to predict the unknown MDP parameters through the framework the predictive model is trained to generate more accurate MDP parameters from the features.

The second step is to use RL on the MDP with parameters to learn an optimal policy. Note, any RL method can be used to generate the optimal policy it is independent to how the optimality of the MDP is calculated. The paper considered two methods for this calculation, namely policy gradients or bellman error. Then the third step is to evaluate the policy using a OPE method which uses the second input to the framework, the episodes generated under the true environment. Any differentiable OPE can be used. From the OPE the error of the generated optimal policy is calculated. The errors are then backpropagated all the way to the model (through the optimal policy) which is used to predict the unknown MDP parameters.

It is important to note, that three sets of data must be kept, one for training, one for testing and one for validation.

1.3.2 Backpropagation Formulation

The actual optimization objective is to find a set of weights of a predictive model ($\theta = \mathcal{F}_w(x)$) that takes in features as inputs and predicts unknown MDP parameter. that maximizes the expected OPE (Eval) metric of the optimal policy ($\pi^*(\theta)$). Note the model is trained on ($\mathcal{D}_{\text{train}} = \{(x_i, e_i)\}_{i \in I_{\text{train}}}$) the OPE uses a set of unseen data which consists of unseen features and episodes ($\mathcal{D}_{\text{test}} = \{(x_i, e_i)\}_{i \in I_{\text{test}}}$). Equation 1 summarizes the objective.

$$\max_{w} \quad \mathbb{E}_{(x,e)\in\mathcal{D}_{\text{train}}}\left[\text{Eval}_e\left(\pi^*(\theta)\right)\right] \text{ with } \theta = \mathcal{F}_w(x) \tag{1}$$

This can be achieved by gradient descent by finding the gradient of the OPE with respect to the weights of the predictive model.

$$\frac{d\,\text{Eval}(\pi^*)}{dw} = \frac{d\,\text{Eval}(\pi^*)}{d\pi^*}\frac{d\pi^*}{d\theta}\frac{d\theta}{dw} \tag{2}$$

In the equation above the term that is hardest to calculate is $\frac{d\pi^*}{d\theta}$. $\frac{d\,\text{Eval}(\pi^*)}{dw}$ depends on the OPE (a differentiable OPE must be used) and $\frac{d\theta}{dw}$ depends on the predictive model used. The full derivation of how the $\frac{d\pi^*}{d\theta}$ is calculated can be found in the paper but the derived equation is shown on equation 3. The summary is that it relies on differentiating the optimality conditions of an MDP. The specific optimality conditions will be further explained later in the report.

$$\frac{d\,\text{Eval}\,(\pi^*)}{dw} = -\frac{d\,\text{Eval}\,(\pi^*)}{d\pi^*}\left(\nabla^2_\pi J_\theta\,(\pi^*)\right)^{-1}\nabla^2_{\theta\pi} J_\theta\,(\pi^*)\,\frac{d\theta}{dw} \tag{3}$$

Note, as mentioned in paragraph 1.2 calculating the hessians can be quite expensive if the action space and state space is large. Therefore, the authors use a sampling technique combined with a low-rank approximation. Unfortunately, due to time constraints (and this being outside the scope of the class) the details to these methods are not covered in this report. The pseudo-code for the actual algorithm is shown below: In the above pseudo-code, line 4 corresponds to a single forward pass

Algorithm 1: Decision-focused Learning for MDP Problems with Missing Parameters

1 **Parameter:** Training set $\mathcal{D}_{\text{train}} = \{(x_i, \mathcal{T}_i)\}_{i \in I_{\text{train}}}$, learning rate α, regularization $\lambda = 0.1$
2 **Initialization:** Initialize predictive model $m_w : \mathcal{X} \to \Theta$ parameterized by w
3 **for** *epoch* $\in \{1, 2, \dots\}$, *each training instance* $(x, \mathcal{T}) \in \mathcal{D}_{train}$ **do**
4 **Forward:** Make prediction $\theta = m_w(x)$. Compute two-stage loss $\mathcal{L}(\theta, \mathcal{T})$. Run model-free RL to get an optimal policy $\pi^*(\theta)$ on MDP problem using parameter θ.
5 **Backward:** Sample a set of trajectories under θ, π^* to compute $\nabla^2_\pi J_\theta(\pi^*), \nabla^2_{\theta\pi} J_\theta(\pi^*)$
6 **Gradient step:** Set $\Delta w = \frac{d\,\text{Eval}_{\mathcal{T}}(\pi^*)}{dw} - \lambda\frac{d\mathcal{L}(\theta,\mathcal{T})}{dw}$ by Equation (8) with predictive loss $\mathcal{L}$ as regularization. Run gradient ascent to update model: $w \leftarrow w + \alpha\Delta w$
7 **Return:** Predictive model m_w.

Figure 2: Pseudo-code for the predict-then-optimize framework applied to sequential decision making

of the framework. Line 5 calculates the hessians required to calculate the gradient. Line 6 calculates the gradient step, the first term in Δw corresponds to the errors evaluated from the final decision and the second term comes from the predictive errors of the model used to predict the unknown mdp parameters.

2 Review of Theory

The following section is a literature review to theory used in the paper with the main goal being trying to bridge the gap between the material taught in class.

2.1 Optimality Conditions in MDPs

As stated in section 1.3.2 to backpropogate the decision errors the term $\frac{d\pi^*}{d\theta}$ is required. This means that the optimal policy (π^*) has to be expressed in terms of the predicted unknown MDP parameters (θ). The authors uses two main methods to obtain this expression, namely through using the bellman error and policy gradient.

2.1.1 Bellman based optimality

The intuition behind the Bellman equation is that the value of a state is derived from considering the immediate reward obtained in that state and the discounted value of being in the next state, ensuring that each step optimally contributes to the overall value. The Bellman error represents the discrepancy between the value estimated by the current model and the value calculated using the Bellman equation. Therefore, a cost function $J_\theta(\pi^*)$ can be give by the mean squared bellman error. The optimal policy using the bellman error can be expressed by the following equation:

$$\pi^* = \underset{\pi}{\operatorname{argmin}}\, J_\theta(\pi) \quad \textit{where } J_\theta(\pi) \doteq \mathbb{E}_{e\sim\pi,\theta}\frac{1}{2}\delta^2_\theta(e,\pi) \tag{4}$$

$$\delta_\theta(e,\pi) = \sum_{(s,a,s')\in e} Q_\pi(s,a) - R_\pi(s,a) - \gamma\mathbb{E}_{a'\sim\pi}Q_\pi(s',a') \tag{5}$$

2.1.2 Policy Gradient

The policy gradient is simply aims to maximize the cumulative rewards under a given policy which can be expressed as the total discounted rewards for an episode ($G_\theta(e)$). The optimal policy, given a cost function ($J_\theta(\pi^*)$), can be given by the following equation.

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, J_\theta(\pi) \quad \textit{where } J_\theta(\pi) \doteq \mathbb{E}_{e\sim\pi,\theta}G_\theta(e) \tag{6}$$

2.2 Off-policy Policy Evaluation (OPE)

Off-policy policy evaluation is a sub-topic in reinforcement learning that aims to evaluate the quality of a policy using a different policy. OPE is largely used in off-line settings where a large amount of historical data already exists. Examples include healthcare and recommendation systems.

The specific OPE used by the authors was the Consistent Weighted Per-Decision Importance Sampling (CWPDIS). A potential problem of importance sampling (IS) is that the variance can be quite high as it is simply a ratio of two distributions. Possible solutions would be to have weighted importance sample or have a per-decision importance sample. The intuition behind per-decision importance sampling is that a different importance weight for each reward at each state rather than for a whole episode. In weighted importance sampling, the original importance sampling values are simply divided to by the sum of importance weights. CWPDIS combines these two notions to obtain an estimation, the value estimate is shown in equations 7 and 8.

$$V^{\mathrm{CWPDIS}}\left(\pi_\theta\right) \triangleq \sum_{t=1}^{T} \underbrace{\gamma^t \frac{\sum_{n\in\mathcal{D}} r_{nt}\rho_{nt}\left(\pi_\theta\right)}{\sum_{n\in\mathcal{D}}\rho_{nt}\left(\pi_\theta\right)}}_{\text{per-decision IS}} \tag{7}$$

$$\rho_{nt}\left(\pi_\theta\right) \triangleq \prod_{s=0}^{t} \underbrace{\frac{\pi_\theta\left(a_{ns} \mid o_{n,0:s}, a_{n,0:s-1}\right)}{\pi_{\mathrm{beh}}\left(a_{ns} \mid o_{n,0:s}, a_{n,0:s-1}\right)}}_{\text{weighted IS}} \tag{8}$$

Note, with respect to the per-decision importance sample if all the rewards before the terminal The OPE metric uses CWPDIS in combination with an effective sample size term to encourage a larger sample size. In equation 9, λ_{ESS} is a hyper parameter.

$$\text{Eval} \doteq V^{\mathrm{CWPDIS}}(\pi_\theta) - \frac{\lambda_{ESS}}{\sqrt{ESS(\pi_\theta)}} \tag{9}$$

$$ESS(\pi_\theta) \doteq \sum_{t=1}^{N} \frac{(\sum_i \rho_{itf}(\pi_\theta))^2}{\sum_i \rho^2_{it}(\pi_\theta)} \tag{10}$$

3 Experiments

3.1 Brief summary and discussion of experiments from paper

Three experiments were performed in the paper, one experiment had unknown rewards and the two other experiments had unknown transition probabilities.

For the experiment with unknown rewards, a 5x5 gridworld setting was used where 80% of the states had a chance of reward following a normal distribution $\mathcal{N}(0, 1)$. The other 20% had a chance of a negative penalty of $\mathcal{N}(-10, 1)$. The starting state is set to the bottom left and the goal state is set to the top right. A random feature vector is generated and correlates to the reward value of the state.

The second experiment was called snare-finding where the transition probabilities were unknown. In this experiment there are 20 states, 4 states have a higher chance of having a snare ($\mathcal{N}(0.8, 0.1)$), whereas the other 16 states have a small chance of having a snare ($\mathcal{N}(0.1, 0.05)$). At each time step the agent has a choice to pick a state, if the state has a snare then the agent will receive a reward of 1 but if no snare is found then the agent will receive a penalty of -1. The fact that the agent does not know whether the state has a snare makes this environment partially observable. Each episode is run for a set number of time steps. If a snare is placed on a state it will remain until the agent visits it making it a sequential decision problem rather than a multi-armed bandit problem. Features are generated for each state and correspond to the transition probability for each state.

The third experiment was called Tuberculosis Adherence, and is somewhat similar to the second experiment. In this experiment, the transition probabilities for each state is unknown. There are 5 states which represent patients. Each patient has a chance to not adhere to their treatment and at each time step the agent can visit a patient and make them adhere. The rewards at each time step is the number of patients adhering. Again, similar to the previous experiment this is unknown to the agent therefore making this environment partially observable. Although also synthetically generated, the features in this environment correspond to information regarding the patient. Figure 2 shows the average and standard deviations of the OPE

	Gridworld		Snare		Tuberculosis	
Trajectories	Random	Near-optimal	Random	Near-optimal	Random	Near-optimal
TS	−12.0±1.3	4.2±0.8	0.8±0.3	3.7±0.3	35.8±1.5	38.7±1.6
PG-Id	−11.7±1.2	5.7±0.8	−0.1±0.3	3.6±0.3	38.4±1.5	40.7±1.7
Bellman-Id	−9.6±1.4	4.6±0.7	0.7±0.4	3.6±0.3	39.1±1.7	40.8±1.7
PG-W	−11.2±1.2	5.5±0.8	1.2±0.4	4.8±0.3	38.4±1.5	40.8±1.7
Bellman-W	−11.3±1.4	4.8±0.8	1.5±0.4	4.3±0.3	38.6±1.6	41.1±1.7

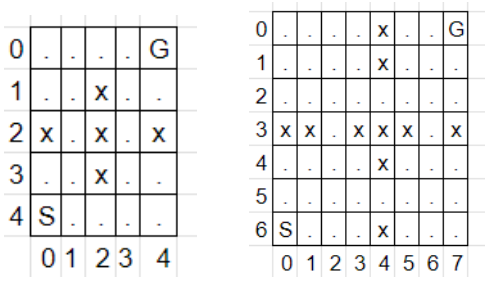
Figure 3: OPE metric values from experiments in the Learning MDPs from Features Paper

metric over 30 runs for the author's experiments. Two types of trajectories were used to evaluate the final policy, random and near optimal (meaning that a policy was learned on the true environment). TS stands for the 'two-stage' which represents the learning the MDP parameters with only the losses from the predictions and without the decision evaluation. PG stands for policy gradient and Bellman stands for the bellman error which correspond to the method used for the optimality of the mdp. Id and W represent the numerical method used to approximate the Hessian. The The results show that for two out of the three experiments the TS and for all experiments some variation of the predict-then-optimize framework does the best. Also, when trained using near-optimal trajectories the framework produces better results.

3.2 Implementation of paper

Several experiments were completed to further test the framework. These experiments were specifically meant to test how important the features were to the success of the framework. One small gripe I had with the author's experiments was that the features that were used were not intuitive. For their experiment each of the unknown values (either state reward or transition probability) were fed into a multi level perception model with a fully connected hidden layer and a forward pass was performed to generate 16 features.

For my implementation two different gridworlds environments were used, one was a 5x5 grid with walls and another one was a 8x8 grid with walls, various features and reward schemes were generated using these environments. In this experiment the rewards for each state was unknown. Each experiment was run 5 times to obtain a mean and standard deviation. Although, not shown here, all the experiments were able to find the optimal path however, this is likely due to the action and state spaces being relatively small.



(a) 5x5 GridWorld (b) 7x7 GridWorld

Figure 4: Gridworld Environments

To simplify the experiment only one feature was assigned to each state. An optimal policy was trained on the real environment using Q-learning to generate the trajectories for training, testing and validation. Q-learning was also used to generate the policy for OPE evaluation. The first experiment was to see what if a desired path was highlighted using the features and rewards, note random noise is added to the reward values in the figure with $\mathcal{N}(0, 1)$. This experiment was done in the 5x5 gridworld environment. The rewards were also used as the features for this experiment and is shown in figure 5, while figure 6 shows the results over 15 epochs.

-5	-5	-5	7	10
-5	-5	x	6	-5
x	-5	x	5	x
-5	-5	x	4	-5
S	1	2	3	-5

Figure 5: Rewards and Features for 5x5 Gridworld with desired path

(a) Test Loss (b) Average Reward Per Episode (c) OPE Evaluation

Figure 6: Test Results for following desired path in 5x5 Gridworld

The first graph is the test loss which represents the difference in reward for a set of trajectories found using the final policy, this essentially indicates how well the model was able to predict the unknown parameters. The second graph (test strict evaluation) is the average rewards per episode for 50 episodes and $\gamma = 0.95$ and the last graph (test soft evaluation) is the OPE evaluation (CWPDIS compared to a set of unseen trajectories from the true environment). The test loss decreases dramatically after a few epochs and the average rewards and OPE evaluation do not change much as the epochs increase.

Another test was performed to see how sparse rewards affect the framework, in this experiment only the goal state had rewards. The features values for were randomly generated with a normal distribution of $\mathcal{N}(5, 5)$ except for the goal state which had a $\mathcal{N}(10, 1)$. The features and rewards are shown in figure 7, while figure 8 shows the results. In figure 7a, the orange shaded area implies that the features are random.

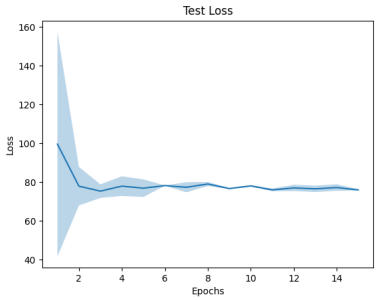
11	13	4	13	9
6	2	x	3	15
x	2	x	11	x
2	3	x	9	3
S	10	3	15	15

(a) Features

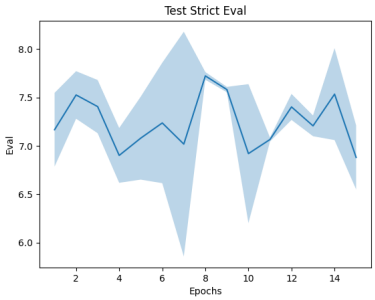
0	0	0	0	10
0	0	x	0	0
x	0	x	0	x
0	0	x	0	0
S	0	0	0	0

(b) Rewards

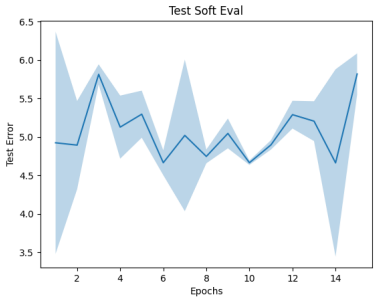
Figure 7: Features and rewards for 5x5 GridWorld with sparse rewards



(a) Test Loss



(b) Average Reward Per Episode



(c) OPE Evaluation

Figure 8: Test Results for sparse rewards in 5x5 Gridworld

In this experiment, the test loss varies more and so does the average rewards per episode and OPE evaluation. The values for the average reward and OPE metric are lower compared to the first experiment due to there being less rewards. The values for test loss is much less than in the first experiment, implying that the model that is used to predict the rewards at each state is closer to the real environment than the first environment.

A potential issue with the above experiments are that the environment is too small so the framework learns the policy very fast and therefore unable to show much improvement as the number of epochs increase. Therefore similar experiments were repeated for a larger gridworld environment. Figure 9 shows the features and rewards in a girdworld of 8x8. Once again, random features were used for all states with 0 rewards and the goal state had reward of $\mathcal{N}(10, 1)$.

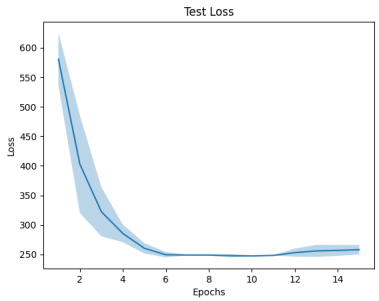
4	14	12	11	x	1	0	10
6	15	7	12	x	0	5	10
2	7	6	0	0	11	2	2
x	x	9	x	x	x	2	x
10	8	2	5	x	1	15	10
15	7	1	9	5	12	10	1
S	12	12	0	x	13	1	3

(a) Features

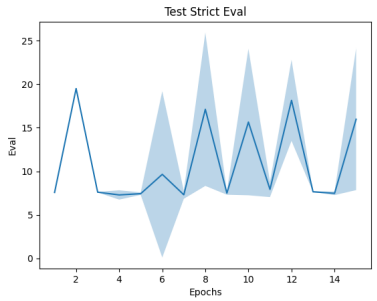
0	0	0	0	x	0	0	10
0	0	0	0	x	0	0	0
0	0	0	0	0	0	0	0
x	x	0	x	x	x	0	x
0	0	0	0	x	0	0	0
0	0	0	0	0	0	0	0
S	0	0	0	x	0	0	0

(b) Rewards

Figure 9: Features and rewards for 5x5 GridWorld with sparse rewards



(a) Test Loss



(b) Average Reward Per Episode

(c) OPE Evaluation

Figure 10: Test Results for sparse rewards in 5x5 Gridworld

Similar to the first two experiments, the framework learns very fast. However, the average rewards seem to not improve that much after each epoch and the OPE evaluation also suffers from the same issues.

Another test was performed with the upper right quadrant also having rewards of an average of 3. This is also reflected in the features.

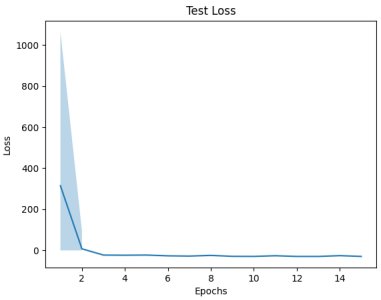
2	10	6	2	x	3	3	10
9	2	2	5	x	3	3	3
8	9	3	1	2	3	3	3
x	x	7	x	x	x	7	x
4	3	6	4	x	11	12	6
3	2	8	12	9	0	8	3
S	7	4	11	x	0	15	15

(a) Features

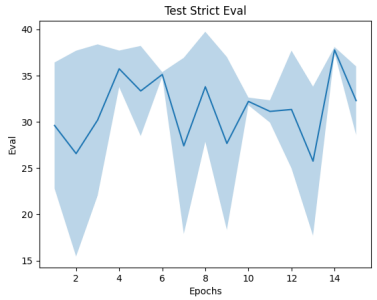
0	0	0	0	x	3	3	10
0	0	0	0	x	3	3	3
0	0	0	0	0	3	3	3
x	x	0	x	x	x	0	x
0	0	0	0	x	0	0	0
0	0	0	0	0	0	0	0
S	0	0	0	x	0	0	0

(b) Rewards

Figure 11: Features and rewards for 5x5 GridWorld with sparse rewards



(a) Test Loss



(b) Average Reward Per Episode

(c) OPE Evaluation

Figure 12: Test Results for sparse rewards in 5x5 Gridworld

Interestingly, in this experiment the test loss was very low after the first few epochs. The average rewards and OPE evaluation slightly increased but not by a significant margin. However, when compared to the previous test, the average reward and OPE metric increases by a relatively significant just from adding more rewards.

3.3 Discussion of results and framework

In hindsight, probably more thorough tests could have been done to better understand this framework. However, due to time constraints this was not possible. The test loss is useful in understanding the accuracy of the prediction but it can be hard to understand how to improve as there are many places where the errors can come from due to the many models used. It does seem that not having sparse rewards helps the framework as shown in the last two experiments. It could also depend on the type of OPE metric being used. In this case since a per-decision metric is used having all the rewards at the end may not be that beneficial. More testing could have been done under larger state and action spaces. I believe this is where this framework will benefit more.

4 Conclusion

4.1 Summary

In this report the paper Learning MDPs from Features: Predict-Then-Optimize for Sequential Decision Making by Reinforcement Learning was studied. In this report, the theory used from the paper was studied and explained. Several experiments were performed using in attempt to better understand the framework.

Also, as this project was somewhat theoretical project, I would feel comfortable giving a 1 hour presentation on this topic. I have also submitted a presentation along with the report.

4.2 Reflection and possible future work

Unfortunately, the project scope had to be changed midway. The original plan was to apply this framework to a medical setting in which the patient would be the MDP enviroment and the doctor would essentially act as the agent. The features would represent the diagnosis from the doctor and since many hospital already track patients progress this information could be used as the existing episodes. Some initial work was put into developing this environment but there was some issues with designing the features and trying to create a simple model. This was because if the model was too simple it would essentially have become a multi-armed bandit problem (i.e. just select the correct treatment for the identified disease). With this said I still believe that this framework would be applicable to this use case as data sets for modelling patient treatment as MDPs do exist and there is a decent amount of literature applying MDPs to healthcare settings.

5 References

[1] Kai Wang, Sanket Shah, Haipeng Chen, Andrew Perrault, Finale Doshi-Velez and Milind Tambe. Learning MDPs from Features: Predict-Then-Optimize for Sequential Decision Problems by Reinforcement Learning, 2021; arXiv:2106.03279. https://arxiv.org/pdf/2106.03279

[2] Adam N. Elmachtoub and Paul Grigas. Smart "Predict, then Optimize", 2017; arXiv:1710.08005. https://arxiv.org/pdf/1710.08005

[3] Joseph Futoma, Michael C. Hughes and Finale Doshi-Velez. POPCORN: Partially Observed Prediction COnstrained ReiNforcement Learning, 2020; arXiv:2001.04032. https://arxiv.org/pdf/2001.04032

[4] Masatoshi Uehara, Chengchun Shi and Nathan Kallus. A Review of Off-Policy Evaluation in Reinforcement Learning, 2022; arXiv:2212.06355. https://arxiv.org/pdf/2212.06355

[5] Phillip S. Thomas. Safe Reinforcement learning, 2019 (Doctoral Thesis). https://people.cs.umass.edu/ pthomas/papers/Thomas2015c.pdf