Preference Learning With APReL Library

Fareed Hassan Khan
MS – Data Science
Institute of Business Administration
Karachi, Pakistan
f.khan.25367@khi.iba.edu.pk

Syed Asad Rizvi
MS – Data Science
Institute of Business Administration
Karachi, Pakistan
s.rizvi.25365@khi.iba.edu.pk

Abstract— **Human-robot interaction has long been an issue in reinforcement learning with respect to the learning of rewards, based on what the user wants the robot to do. In such a scenario, preference learning phenomenon comes up with a solution that is based on learning from human preferences rather than reward signals. One such library, called APReL is proposed and replicated in this paper that contains various active preference-based reward learning techniques. A classic control environment of "Mountain Car" is used along with its features, whose values are used to identify the minimum and maximum position of the car, along with its velocity. Preference learning is used in this environment to make the car reach its goal state at the top of the right hill. A number of trajectories are generated during training, which in turn asked for human preferences based on the number of queries and trajectories provided. The algorithm contains different parameters that control the features of the car which have produced the results accordingly. This analysis would fit in similar kinds of robotic applications where the objective is to control and learn from human preferences. In future, these techniques can be extended to other learning methods such as neural networks and gradient-based algorithms.**

I. Introduction

Preference learning is a sub-domain of Reinforcement Learning which centers around learning from human preferences rather than reward signals. Learning from preferences is a fundamental task here, which requires a pair of trajectories or actions to be presented to users, who choose their preferences accordingly. Estimating a reward function by incorporating these preferences is the main objective in preference learning. This allows the interaction between agent and environment to be more effective, hence making it very useful for real-world problems, particularly in the field of robotics [1].

Preference learning offers various solutions to RL problems and challenges. First and foremost, it is eliminating the need of manually specifying a reward function which is problematic in a sense that it could be uncertain, erratic, and time-consuming. Secondly, in complex domains and environments, the agents tend to perform ineffectively due to high ambiguities. Also, learning optimal policies in the presence of human feedback is more recommended instead of learning with delayed rewards with manual reward functions.

Such a problem is studied in this project that contains a "Mountainous Car" environment, whose objective is to accelerate the car from the bottom of a valley, such that it reaches the goal state on top of the right hill. Using preference learning, the process involves selecting the best option by human users from a pair of trajectories obtained from the environment. Based on the user preferences, the agent gets to know which trajectories to take, so that it can learn such policies that produce a successful output, reaching the goal state of Mountain Car environment in this scenario.

In this paper, we have replicated a novel Python library called APReL that contains various active preference-based reward learning techniques. The challenges presented by preference learning involve producing a little amount of information from the pairwise trajectories. APReL library provides such methods and learning techniques which improves the efficiency of the process by maximizing the information obtained from user queries, hence overcoming the challenge. This library is applicable for any environment which is adaptable to OpenAI Gym [1].

II. Related Work

A. Methodology

We chose to select 12 papers for summarization based on their relevance and quality after conducting a thorough literature review on preference learning. Our goal was to gain a comprehensive understanding of the field and explore the various methodologies and approaches being used in those papers. Through this selection process, our aim was to provide a comprehensive overview of preference learning research and its potential implications.

One particularly notable paper among our chosen selection is "A Library for Active Preference-based Reward Learning Algorithms (APReL)." We considered this paper for its significant contribution to the field of active preference learning and its potential impact on practical applications.

The availability of code on GitHub further enhanced its value, as it provided a valuable resource for implementation and experimentation. By including this paper, we sought to explore the advancements in active preference-based reward learning algorithms and their potential applications across diverse domains.

The selection of this specific paper aligns with our objective to highlight the importance of active preference learning. By featuring a library that focuses on active preference-based reward learning algorithms, we aimed to showcase the practicality and potential benefits of incorporating user preferences into reinforcement learning frameworks.

This collection highlights the importance of actively learning preferences to improve performance and customization, and its relevance in the broader field of reinforcement learning.

B. Pros and cons

The APReL library offers several advantages to researchers and practitioners in the field. Firstly, APReL provides a comprehensive collection of existing techniques and algorithms, allowing users to experiment with different approaches and easily develop their own algorithms for various aspects of the problem. This accessibility and flexibility enable users to utilize the algorithms to their specific needs and research goals. Moreover, APReL is written in Python, a popular programming language in the machine learning and robotics communities, making it easily integrable into existing workflows.

Another advantage of APReL is its focus on active learning, which maximizes the information acquired from each query to the user, thereby improving data efficiency. By intelligently selecting queries that are most informative, APReL optimizes the learning process and reduces the number of queries needed to learn the reward function accurately [1].

This active learning capability is particularly important in preference-based learning, where each comparison or ranking provides a small amount of information. APReL's active learning techniques allow users to make the most of the limited information available, enabling faster and more efficient learning of human preferences.

However, APReL also has certain limitations and considerations. One potential drawback is that preference-based learning relies on human input, which can introduce subjectivity and variability. The accuracy and reliability of the learned reward function heavily depends on the quality of the user's preferences and rankings [1].

Therefore, ensuring the user's understanding of the task and providing clear instructions are crucial for obtaining meaningful results. Additionally, APReL's effectiveness might be limited in scenarios where demonstrations or observations are difficult to collect or unreliable. Users should carefully consider the suitability of preference-based learning and explore alternative methods depending on the specific application and available resources.

III. METHODS

In this section, we have discussed the methods and parameters given by APReL library, what each hyperparameter is doing to the process, their tuning, and effect of different values.

a. *Number of Trajectories:*

It generates the number of trajectories as desired by the user, and based on that, preferences (A and B) are created afterwards. Its default value is 40, means at least 40 trajectories are created in each iteration every time.

b. *Maximum Episode Length:*

It tells the maximum length of an episode, particularly representing how many state action pairs are in each episode. It shows the maximum number of time steps per episode only for the new trajectories.

c. *Number of Samples:*

This parameter represents the number of sample preferences or pairwise trajectories that a user provides. Its default value is 100, and we tune it to lower or higher values to obtain different results.

d. *Query Type:*

There are several query types that is asked from the user to select, based on the number of trajectories:

Preference: It is the default value of query type parameter. It simply asks the users which trajectory option they prefer (A, B, C etc.).

Weak Comparison: It gives the user the option to select whether a trajectory is normal or indifferent i.e., when the user is unable to decide which option to choose.

Full Ranking: In this type, we can rank our trajectories and decide which option to choose based on this ranking. It is mostly used when we have more than 02 trajectories, so we can rank them based on observation and choose the best trajectory.

e. *Query Size:*

It tells how many numbers of queries are to be generated for preferences. For example, query size of 4 means four option of trajectories to choose from (A, B, C and D).

f. *Number of Iterations:*

It specifies the number of times asking the user which trajectory to choose. Its default value is 10, and we can tune it according to our requirements.

g. *Optimization Method:*

This parameter represents the optimization method used to find the reward function based on preferences. They are basically used for generating batches of queries. It contains the following methods:

Greedy: A batch generation method which selects some queries from a particular batch of reduced set of queries with respect to the acquisition function used, to maximize the reward based on their differences.

Medoids: This method uses the same technique as centroids in k-means clustering algorithm. It creates clusters of reduced set of queries using k-medoids method, that depends on a given distance measure.

Boundary Medoids: Like medoids, this method applies k-medoids to the reduced set of queries, but now on the boundary of the convex hull of the reduced set. The limitation of this method is that it can only be used when the queries are mapped in Euclidean space, due to the unavailability of boundaries of convex hull most of the time.

Successive Elimination: It follows an elimination rule in which several queries are removed iteratively from the reduced set. The rule states that the pair of queries with the shortest distance having the worst acquisition function value are eliminated from the set.

Determinantal Point Processes (DPP): This method uses the distance between queries and acquisition function values to maximize the diversity of the set of queries for the batch.

h. Batch Size:

Batch size parameter specifies how many batches of generated queries we are selecting at a time, needed to learn the reward function.

i. Acquisition Function:

These functions represent actively generating queries instead of batches, which are required in the overall process.

Volume Removal: It is the kind of acquisition function in which the objective is to remove the maximum volume from the belief distribution. It searches those queries that show the uncertainty about the preferred human behavior [2, 3].

Disagreement: It queries the user with trajectories that are likely similar. Its approach is to maximize both likelihood and disagreement of two estimates. The optimal disagreement query after passing these values. A planner is used here which provides the optimal trajectories based on the estimated values [4].

Mutual Information: This parameter maximizes the information that will be gained from the user's response. It provides improved learning efficiency as compared to the previous two parameters because they also allow "weak comparison queries" to be the part of learning [5].

Regret: It queries the trajectories that are likely to incur high regret with respect to each other. It is a bit like disagreement parameter in the sense that it also uses a planner that provides optimal trajectories. Although here, the goal is to find optimal behavior rather than the reward function [6].

Thompson Sampling: It uses samples from the belief distribution to generate the query. The optimal trajectory is quickly learnt in this method because it minimizes the regret during querying [7].

The choice of acquisition function depends on query type, the task, and the overall objective.

j. Restore

This command is used for restoring the previous trajectories in case we don't want to generate new trajectories. We can resume from where we last stopped the trajectories, using this parameter.

k. Headless:

No visualizations will be shown if we turn the headless mode on. In this way, we can observe our results from the weights array generated after each trajectory.

IV. EXPRIMENTAL RESULTS

This section explains tuning of hyperparameters to explore their impact on the Mountain Car environment. By examining the experimental outcomes, we gain valuable insights into how hyperparameter tuning influences the Mountain Car environment. Moreover, we conducted a replication study of the original paper's hyperparameter tuning process to further explore its impact on the Mountain Car environment.

i. Replication:

In the replication of the paper's results, the same feature function was designed, and the experiment settings were set as follows: using random trajectories (10 in total), a maximum episode length of 300 to restrict trajectory length, a seed value of 0 for consistency, and 10 iterations of querying. The learned policy weights obtained were not significantly different from those reported in the paper, but there were some variations. The specific differences between the estimated weights are as follows: the first weight, which corresponds to the minimum position of the mountain car, was estimated as -0.3344529 in our replication, while the paper reported a value of -0.28493522. For the second weight, representing the maximum position, our replication yielded 0.80711908, whereas the paper reported 0.72942661. Lastly, the third weight denoting the mean speed was estimated as 0.4865183 in our replication, whereas the paper indicated a value of 0.62189126.

ii. Hyperparameter Tuning

The experimentation starts with generating 200 random trajectories and utilizing an active learning loop with a maximum of 10 iterations. Despite the numerous trajectories, the car falls short of reaching the flag, only achieving a maximum distance of 0.854. However, it becomes evident that reaching the flag is possible based on the random trajectories. To improve the results, we increased the number of learning loops. Subsequently, in a new experiment with 200 random trajectories and a maximum of 20 iterations, the active learning loop is extended. At the 15th iteration, the car demonstrates progress, achieving a maximum distance of 0.980. Although this shows improvement, the car still does not manage to reach the flag.

Additional parameters were tuned alongside the previous ones to enhance the performance of the car.

a. Adjusting Max Episode Length

By setting the max episode length parameter to 1000, while keeping the number of trajectories at 100, the number of iterations at 10, the query size at 5, and running the experiments in headless mode, the car achieved a promising maximum position of approximately 10.866 on the 5th query.

This indicates that allowing longer episodes during training enabled the car to make significant progress towards reaching the flag.

However, when the max episode length was reduced to lower values of 200, 300, and 500, the car's performance suffered. These shorter episode lengths were not sufficient for the car to reach the maximum position or ultimately reach the flag. It suggests that longer episodes provide more opportunities for the car to explore and learn from its actions, leading to better overall performance. Thus, the findings highlight the importance of carefully tuning the episode length parameter to optimize the car's learning process and improve its chances of successfully reaching the flag.

b. Adjusting Query Types

Initially, with 40 trajectories and 10 samples, the weak_comparison parameter did not have a significant impact on reaching the flag. Both trajectories appeared the same, resulting in a value of -1 being passed. The maximum distance between the trajectories was 0.518 with the weak_comparison parameter and 0.556 without it. Thus, the addition of the weak_comparison parameter did not greatly affect the car's ability to reach the flag.

Next, the number of trajectories was increased to 50, and the number of samples was increased to 20. At the third sample, both trajectories still looked the same, leading to a value of -1 being passed. The maximum distance between the trajectories with the weak_comparison parameter was 0.937, compared to 0.556 without it. Increasing the number of trajectories and samples had a significant positive impact on the car's performance in reaching the flag.

Following that, the number of trajectories was further increased to 150, and the number of iterations was set to 20. This adjustment resulted in a higher maximum distance of 0.995 between the trajectories. Despite the increased distance, the car was still unable to reach the flag, indicating that further improvements were necessary.

Continuing the experimentation, the number of trajectories was set to 200, and the number of iterations was increased to 40. A maximum distance of 0.986 was achieved, despite the car making some progress and getting closer to the flag, it still failed to reach the final spot. It is important to note that when the query size and max episode length were tuned, the car was able to successfully reach the flag.

After the previous approach of using weak comparison, the parameters were tuned to new values. With 100 trajectories and 30 iterations and using the full_ranking query type this time; the car successfully reached the flag at one point during the random generation phase. However, when preference selection was applied, the car's performance did not improve, as it failed to reach the flag within the 30 iterations. The maximum position achieved was 0.77, which was lower than the result obtained with the weak_comparison query type.

The active learning loop was modified by increasing the number of trajectories to 200 and setting a maximum of 40 iterations, keeping the query type value same (i.e., full_ranking). The experiment did not yield any improvement in reaching or getting close to the flag. The car's performance remained limited, with a maximum distance of 0.78, showing no progress compared to the previous experiment.

c. Modifying Query Size

In all the previous experiments, the query size parameter was not tuned individually. To explore the potential benefits of adjusting the query size, we conducted further experiments:

By setting the -num_trajectories to 100, num_iterations to 10, and query_size equal to 5, we observed that the car successfully reached the flag at query #4. In headless mode, the car's maximum position was approximately 10.866. This result implies that increasing the number of queries could potentially enhance the car's ability to achieve the goal.

Increasing the number of trajectories to 200, raised the iteration count to 30, and utilized a query size of 20. Fortunately, the car managed to reach the flag but at a later query, this did not lead to improved performance. This experiment is done just to check the relationship between these two parameters with respect to our goal. Additionally, it is worth mentioning that the computation time increased significantly, by up to 100%, due to these parameter adjustments, compared to our previous findings.

d. Altering Batch Size

A new parameter, "batch_size," was introduced to evaluate further analysis. The experiment was conducted with the following parameter values num_trajectories with value of 100, num_iterations upto 10, query_size 5 and batch_size equal to 3. During the evaluation in headless mode, the car achieved a maximum position of approximately 10.866 on query #3 and #5 with the given parameter values. This indicates that the car's performance remained consistent with the previous experiment.

Further observations were made by increasing the batch size to 5 and 10 while keeping the other parameters the same. However, it was noted that these adjustments did not lead to any improvement in the algorithm. The car's maximum position remained unchanged, indicating that increasing the batch size did not significantly affect the car's ability to reach the flag.

e. Adjusting Acquisition Functions

In the exploration of acquisition function parameters for a car's navigation, different values were tested for the parameters: mutual_information, volume_removal, disagreement, regret, and Thompson sampling. For the first test using mutual_information, the car successfully reached the flag within the first two queries.

Next, the parameter was changed to volume_removal while keeping other parameters constant. It was observed that the car was able to reach the flag at every odd-numbered query, indicating a slower but consistent progress towards the goal.

Then, the parameter was set to disagreement. In this case, the car's maximum position reached was recorded at 10.866, and the flag was successfully reached at every query, demonstrating a consistent and effective strategy.

Similarly, when the parameter was changed to Thompson sampling, the car also reached the flag at every query, indicating a reliable and successful navigation strategy.

f. Adjusting Optimization Methods

Different optimization methods were experimented with in the parameter tuning study while keeping the trajectories and iterations constant. Exhaustive search, greedy, medoids, boundary medoids, successive elimination, and DPP were among the optimization methods evaluated. Surprisingly, regardless of the method used, the car consistently achieved a maximum position of around 10.866.

This experiment revealed that the choice of optimization method had little influence on the outcome. Regardless of whatever optimization method was used during the experiment, the car consistently achieved a maximum position of around 10.866 on the 5th query in headless mode.

V. CONCLUSION

In this project, a Python library called APReL was introduced, which focuses on preference-based learning. The library offers a modular framework that allows for experimentation and development of active preference-based reward learning algorithms. Throughout the project, various techniques and features were discussed. To evaluate the effectiveness of APReL, experiments were conducted using the OpenAI gym mountain car environment. The results showed that the implementation of APReL had an impact on the outcomes, enabling the car to reach the flag in earlier queries in some cases but not in others.

VI. REFERENCES

[1] Erdem Bıyık, Aditi Talati, and Dorsa Sadigh: APReL. A Library for Active Preference-based Reward Learning Algorithms: 2022, https://arxiv.org/pdf/2108.07259.pdf

[2] Dorsa Sadigh et al. “Active Preference-Based Learning of Reward Functions”. In: Proceedings of Robotics: Science and Systems (RSS). July 2017. DOI: 10.15607/RSS.2017.XIII.053.

[3] Erdem Biyik et al. “The Green Choice: Learning and Influencing Human Decisions on Shared Roads”. In: Proceedings of the 58th IEEE Conference on Decision and Control (CDC). Dec. 2019. DOI: 10.1109/CDC40024.2019.9030169.

[4] Sydney M Katz, Anne-Claire Le Bihan, and Mykel J Kochenderfer. “Learning an urban air mobility encounter model from expert preferences”. In: 2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC). IEEE. 2019, pp. 1–8.

[5] Erdem Biyik et al. “Asking Easy Questions: A User-Friendly Approach to Active Reward Learning”. In: Proceedings of the 3rd Conference on Robot Learning (CoRL). Oct. 2019.

[6] Nils Wilde, Dana Kuli´c, and Stephen L Smith. “Active preference learning using maximum regret”. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE. 2020, pp. 10952–10959.

[7] Maegan Tucker et al. “Preference-based learning for exoskeleton gait optimization”. In: 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE. 2020, pp. 2351–2357.