Practicum Report

Reinforcement Learning applied to Digital Marketing

Anushital Sinha

OMSA, Georgia Institute of Technology

MGT 6748, Applied Analytics Practicum, Summer 2022

Preface

I did my applied analytics practicum at Moksha.ai under the supervision of Mr. Nabeel Siddiqi. My motivation for this project was to give the consumers a unique and customized experience during their buyers' journey and to help digital marketers to be able to do so in a cost effective manner.

The **goal** of this project was to build a proof of concept reinforcement learning model to predict the best digital ad-website combination to run at any given point in time.

My **duties** included building the proof of concept model, building the simulation environment for testing, interviewing the subject matter experts and determining which assumptions to include in the modeling.

Overall, my time in this project was intense and challenging but very fruitful. I learned about a myriad of cutting edge technical topics and details of the inner workings of digital advertising. I began this semester by learning Reinforcement Learning. I started with OpenAI's Spinning Up, which was useful as an intro but quickly things went too deep too fast so then I moved on to the textbook, Introduction to Reinforcement Learning by Sutton and Barto and the first few chapters of this book gave me enough understanding of RL for my project. My supervisor told me in the beginning of the semester that I needed the concepts of probabilistic programming for Agent model application in the project. So at this point, I started going through PyMC's tutorials and documentation, which is a good introduction on probabilistic programming. Then I moved to reading Statistical Rethinking which was an excellent resource for understanding, but didn't provide the syntax translation that was required in the PyMC model used to implement Bayesian Linear Regression used in this project. So I picked up the book Bayesian Methods for Hackers by Cameron Davidson - Pilon which has some really good PyMC examples. I read the PyTorch documentation for deep learning basics and how to use distributions inside a neural network.

During the entire phase of working on the project hands-on, I had my fair share of challenges and frustrations. Implementation of PyMC bayesian linear regression did not have an easily implementable update method. The implementation of the agent was object oriented but the PyMC code examples are not, adapting PyMC code to OOP was difficult. I explored Bambi, a library based on PyMC but with a simplified interface, however it did not have an update method, hence I came back to PyMC and figured out a workaround of re-training with all the data available.

PyMC was very slow in running both training and inference, so I explored migrating to NumPyro, but did not have enough time to migrate and test the code. I have used Thompson Sampling in this project to predict which is the best ad-website combination. But it was not readily implemented in any library, hence I needed to both understand the math and implement and test the model. PyTorch distributions that are used in Bayesian neural networks seem to not have been used extensively in examples and so finding help on error messages and issues was quite difficult. With much trial and error and investigation and reading PyTorch source code, I was able to solve the problems that I encountered. During all this time, I also had to deal with Apple MacBook M1 compatibility issues which was also a great learning for me.

In terms of my time with the team at Moksha.ai, most of my time was spent interviewing subject matter experts on how the digital marketing space works, simulation environment architecture and default parameters and assumptions to use. I also spent time talking to data scientists on the team to narrow down possible models to use for the agent.

I thank the team at Moksha.ai and my supervisor, Nabeel Siddiqi, for guiding me and answering my never-ending list of questions throughout this semester.

I would also like to thank Dr. Joel Sokol for giving us, the MGT 6748, Summer 2022 batch of the students of OMSA, the opportunity to complete the Applied Analytics Practicum as a part of the course requirement. It was a great learning experience to work on this practicum project.

Lastly, I would like to thank the Teaching Assistants for being a great support throughout this entire semester.

Abstract

This paper is focussed on creating a system to improve the efficiency of digital advertising by building a reinforcement learning based system to dynamically decide on ad creative and website combinations. During the course of the project, a simulation environment was created along the lines of OpenAI's Gym framework with the capability to alter the expected performance by day of the week, week or year and holiday status. For the agent design, three models were tested: Thompson Sampling, Bayesian linear regression and Bayesian neural networks. The results showed that Thompson Sampling performed the best based on minimizing total regret of click-through-rate performance.

Introduction

The invention and subsequent penetration of the world wide web into the electronic gadgets of 5 billion users worldwide, has opened up unprecedented opportunities for digital marketers [10]. The world is getting more digital by the day and it wouldn't be incorrect to label today's consumer as "cyber consumer". These cyber consumers want customization in everything possible like the product and services they want to buy, the information they seek, the ads targeting them and even the prices offered to them. They are competent enough to sort products based on the desired attributes, functionality, prices etc. along with user ratings of these products. " Empowered by digital technology, these customers are unforgiving" [4]. To keep up with these potential customers, the marketers need to improve their marketing strategies accordingly.

The emergence of these cyber consumers has increased the emergence of digital marketing which is more customized and unique to the individual consumer in a cost effective manner:

> In the new environment, marketers are able to consider consumers individually, customize their services and products, and "establish dialogues with consumers" rather than talk "at" them. This is due to the unique and powerful characteristics of the WWW. [11]

The primary goal of the digital marketers is to achieve a high Click Through Rate (CTR) of the ads placed by them on the internet. "For ads that have been displayed repeatedly, this is empirically measurable, but for new ads, other means must be used. For ads that have been displayed repeatedly, this is empirically measurable, but for new ads, other means must be used. We show that we can use features of ads, terms, and advertisers to learn a model that accurately predicts the click-through rate for new ads" [14]. In this project, I have used ad feature in designing the Reinforcement Learning model policy which has improved the CTR prediction of the model.

When an advertiser goes to a web publisher to place an ad, they need to have a prediction of the CTR in place to accurately provide appropriate inventory of ad spots to an advertiser [15]. This can be tricky in case of new ads because they don't have a history from which predictions can be made. In such cases, Reinforcement Learning (RL) seems to be a better choice over Machine Learning Algorithms because the history of the ads is not required for model building in case of RL. Additionally, RL also intelligently explores the data in an efficient way to minimize the amount of useless data that needs to be collected.

Since the goal of any digital marketer is to maximize the CTR, they often face the following three challenges in making advertising decisions :

1. Where to advertise?
    - Youtube, Facebook or Google
2. Which type of ads to use?
    - Video, banner or Audio ads
3. Which creatives to use?
4. How do advertisement consumer time (of the day/week/year) and location affect the above three?

Through my Practicum project, I implemented a solution to the above mentioned challenges faced by the e-commerce businesses using reinforcement learning algorithms and thus find the optimal budget allocation for ad spend.

As per the current practices in the industry, the ROI of ad spend is calculated after the ad campaign ends and then the advertising budget is allocated. But in this whole process, valuable time and resources are wasted, especially the period between the analysis and the reallocation of ad spend. In interviewing subject matter experts, I learnt that money and time are wasted because the budget allocation is almost always not optimal, which results in the marketing budget being allocated for less effective ads leading to less overall efficiency in advertising.

To mitigate this problem, I have attempted to create an automated ad allocation system using Reinforcement Learning where the model keeps learning in real time. The results can be used to reallocate the ad spend, as soon as results come in saving significant time and improving the efficiency of the advertising budget.

Related Work

As per the current practices, the usage of machine learning in digital advertising is primarily focused on the prediction of CTR for ads to determine how many impressions of an ad to show. This is common on the supply side ad platforms, e.g. Facebook may have a similar model for budgeting impressions for an ad [17].

Nonetheless, there has been some research that is more closely related to this project. Multiple papers related to real-time bidding for ad space have been published. In these papers, researchers used a multi-step reinforcement learning model to conduct real time bidding for digital ads [18] [19] [20] [21] [22]. However, the goal here was on matching user interests with the ad content and winning bids rather than determining

which ad performs the best through CTR or conversions. The model setup and training process was informative, but not directly related.

In marketing optimization, the usage of more relevant models to this project are seen. For example, a research paper by the HubSpot team explored the performance of Thompson Sampling and Neural Networks in standard reinforcement learning benchmarks [23]. Though the application of these models was not discussed in the paper, the model formulation was useful in informing approaches that have been used in this project.

Finally, there is one textbook that deals directly with the problem discussed in this project [24], however, it was published in 2011 and the material is dated. In addition, the problems that the authors are solving are: layout of the banner, interactivity of the banner and the timing of delivery of the banner. These are tangentially related to this project and can easily be incorporated into the general framework discussed in the methodology section.

Methodology

This section explains the details of the algorithms and methodology used to implement and test the algorithms and the thinking that went behind the decisions taken.

Reinforcement Learning

The primary method used in my practicum was reinforcement learning. Reinforcement learning is a technique that interacts with an environment and based on feedback from the environment, the algorithm learns and makes better choices. The figure below gives a general idea of how an RL process works:

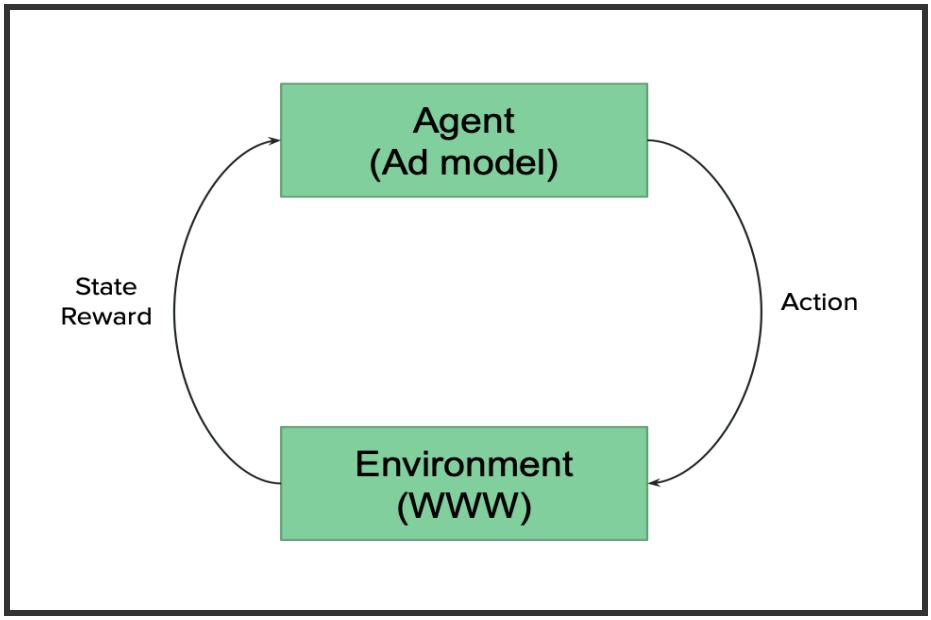

Figure 1: Reinforcement Learning Model

Here the Agent is a model that can be updated incrementally (for the practicum this was a model that chose which ad creative and website combination to use). The environment is the external environment, in this case it is the world wide web. The process unfolds as follows:

1. The environment provides a starting state to the agent
2. The agent predicts the action to take and send it to the environment
3. The environment takes into consideration the state and action and returns a reward and the next state to the agent
4. The agent takes the reward and updates its model and then predicts an action with the next state and provides an action to the environment

For the practicum, one way to model this would be as follows:

1. The environment provides the details of the day on which the ad will be run (this is the state), specifically:
    a. Is the day a holiday
    b. Which day of the week it is - Sunday, Monday etc
    c. Which week of the year it is - holiday week, new years' week or a normal week etc.
2. The agent uses the state to predict the action (in some agents that were tested e.g. Thompson Sampling, the state was ignored, more on this later). For this project the action was a tuple of two values, the first denotes the ad creative and the second denotes the website that the creative will be run on

3. The environment takes the state and action as input and returns the reward, which in this case is a click through rate (CTR) that the ad-website combination achieved. The reward and next state is sent to the agent
4. The agent updates its model with the reward (CTR) and then predicts with the next state the action to take and sends it to the environment. The update of the model depends on the model type being used, in the case of Thompson Sampling it is updation of the Beta distribution parameters, in the case of Bayesian Linear Regression, it is an update of the priors and in Bayesian Neural Networks, it is a backward pass with the new data, more on this below

In the reinforcement learning literature [1], there is a separation between temporal and non-temporal algorithms. Temporal algorithms have a time-based dependency between the decisions that are taken, for example, in chess it is necessary to consider the effect of a move on consequent moves, hence there is a temporal dependency on the actions (an action here is a move in chess) that are being taken. Temporal algorithms can be quite complicated and are at the cutting edge of research [2], however, for this project it was not necessary to use temporal algorithms, though a case for using them can be made. Consequently, the reinforcement learning algorithms used for this project are non-temporal algorithms that assume that all the information required to make a decision is available (this is known as the Markov Assumption [4]). Details of the implications of this decision and other implementation details follow in the next section.

Implementation Details

The implementation of the project had two major components: the environment implementation (representing the world wide web) and the agent implementation (representing the decision making model).

Environment Implementation

Unlike machine learning, reinforcement learning needs a dynamic environment to interact with during the training process. This can get tricky because using a real-world environment can have expensive ramifications (e.g. when the model is exploring the problem space) and can be quite slow since the learning process would be happening in real-time. To solve this problem, the OpenAI group has released an open source library called Gym [5], which was used in this project to simulate the environment.

The environment is setup as a inherited class of gym.Env (full code in appendix) and has the following initialization parameters:

- **Creatives**: a list where each element is the distribution parameters for the CTR for a given creative, these can be thought of as priors for each creative. The

distribution parameters were assumed to be the alpha and beta parameters of a Beta distribution. The number of entries was assumed to be the number of creatives that are being considered

- **Websites**: a list where each element is the distribution parameters for the CTR for a given website, these can be thought of as priors for each website. The distribution parameters were assumed to be the alpha and beta parameters of a Beta distribution. The number of entries was assumed to be the number of websites that are being considered
- **Modifiers**: a dict where each key is either a day of the week, week of the year or a holiday and the value is the distribution parameters for the CTR for the given time period, these can be thought of as priors for each time. The distribution parameters were assumed to be the alpha and beta parameters of a Beta distribution
- **Base distribution**: this is the base distribution that is then modifier, you can think of this as the prior on the whole CTR level of the web e.g. in the early days of the internet the CTR levels may have been higher than modern times due to the noverly of digital ads, this value helps set that assumption, in this case we set it to a neutral (1,1) which refers to a uniform distribution of the Beta distribution

The following details are also useful for understanding:

- The number of actions available is the cartesian product of the creatives and the websites, it is assumed that all actions are available in every state
- The inner details of the environment are not known to the agent, the goal is to see if the agent is able to figure out, via trial and error, which ad-website combination should be run
- The assumption is that CTR ranges between 0 and 10%, hence the output of the distribution is re-scaled for that range
- The distribution parameters for the creatives, websites and modifiers were based on subject matter expert interviews at Moksha

A walkthrough of an example iteration will be helpful for understanding:

1. The environment is initialized and the first state is provided to the agent, e.g. (0,1,2), which means: 0 mean the day is not a holiday, the 1 refers to a Tuesday and the 2 denotes that it is week 3 of the year
2. The agent predicts the best action to take based on the state, if it's a first action it is essentially a random action but with some history the actions become more "intelligent", in this case let's say the agent provides the action (1,0), the 1 denotes ad creative 1 and 0 denotes website 0
3. The environment takes in action and, combined with the state, outputs a reward e.g. 4% CTR. The reward is calculated by the following algorithm:

    a. Based on the "today" date, determine which modifiers are applicable, in this case the state is (0,1,2), hence the applicable modifiers are: Tuesday, Week 3 of the year, let's say the modifiers have the values (2,1) and (0.75,1) respectively
    b. Adjust the base distribution parameters with the relevant modifiers by averaging the parameters e.g. the base is (1,1) and the modifiers are (2,1) and (0.75,1), hence the final Beta parameter will be (1+2+0.75) / 3 = 1.25 and (1+1+1) / 3 = 1 -> (1.25,1)
    c. Sample from the adjusted Beta distribution with parameters (1.25,1) and scale the sampled value between 0% and 10% and return that as the reward to the agent
4. The environment also outputs the next state, along with the reward from 3, the process then repeats from step 2

With a, hopefully, clearer understanding of the environment, it is time to turn to the agent details.

Agent Implementation

The agent is really the core of the system that we are interested in. The agent's job is to predict which ad-website combination to run given a state. The goal is that over time, with experience, the agent will learn what is the best combination given the current state.

There are four agents that have been implemented and tested. The details of the agents are below:

Random Model

This is a simple model intended for baseline measurement, the model predicts an action randomly and does not learn anything at all

Thompson Sampling Model

Thompson Sampling is a technique that uses Beta distributions to determine the probability of an action to be taken. The model learns by manipulating the parameters of the distribution to increase the probability of the actions that result in higher rewards and decreasing the probability of the actions that result in lower rewards. To predict an action, the model samples from each Beta distribution and picks the one with the highest value. This model enables intelligent exploration and exploitation trade-off by exploring more when the performance of an action is unsure and exploiting when there is more confidence in the performance of an action.

Bayesian Linear Regression Model

The Bayesian linear regression model consists of a Bayesian linear regression <ref bishop> for each action possible. This enables each action to take into account the state and remain independent of other actions. This was a design choice, and it is possible to create a single model that incorporates all the actions in one, however with significantly non-linear relationships I chose to go with a model per action.

The prediction process for this model is similar to Thompson Sampling: given a state, each model predicts the expected CTR for the respective action and the highest prediction is chosen. This enables more exploration when the model is uncertain and more exploitation when the model gains confidence.

The theoretical advantage of Bayesian linear regression over Thompson Sampling is that the latter does not consider the state and solely decides actions based on their performance overall. So, for instance, if on the weekend the CTR is markedly higher than on weekdays, Thompson Sampling will not be able to capture this nuance, whereas Bayesian linear regression takes the state into consideration.

In terms of the implementation, the response value is a log-normal distribution since CTR is positive. Nonetheless, due to the fact that Bayesian linear regression is a simple model, a more complex model, capable of handling highly non-linear relationships was also considered.

Neural Network Model

The neural network model, or more specifically a Bayesian neural network <ref BNN paper>, was chosen to see if non-linear relationships could be captured in the data. In this model, similarly to Bayesian linear regression, I have chosen to have a neural network per action, rather than having one large neural network for all actions.

The prediction process is exactly like Bayesian linear regression: given an observation, each action's neural network predicts the CTR and the best one is chosen. The reason Bayesian neural networks are used instead of vanilla neural networks is to enable exploration. This is done by adding a Beta distribution to the final output. This enables exploration of options intelligently, which is required for effective reinforcement learning.

Performance Measurement

To measure the performance of the four agent models (random, Thompson Sampling, Bayesian linear regression and neural networks), the total regret metric was used. This is defined as:

$$Total\ Regret = \sum_{i=0}^{A} R^{*} - R_{i}$$

Where:
A: complete history of the actions taken by an agent
R^{*}: the best reward for the given state
R_{i}: the reward for the action that the agent chose

It is only possible to calculate total regret when we know what the best reward is. In the case of the environment implementation of this project, this was done by simulating each action and then picking the best CTR and returning that as the best reward value.

Results

The results showed that the Thompson Sampling model performed the best out of all the models. Figure 2 depicts the density plot of total regret for each model.

Here Thompson Sampling has the best performance with the random model and neural networks quite close in performance. Further investigation is required, but I would hypothesize that the neural network does not have enough data points to find a good fit.

Another view of the performance of the models is to view the total regret over time, with time being represented by steps (a step is a single iteration of the reinforcement learning cycle, as discussed in the methodology section). This is represented in figure 3.

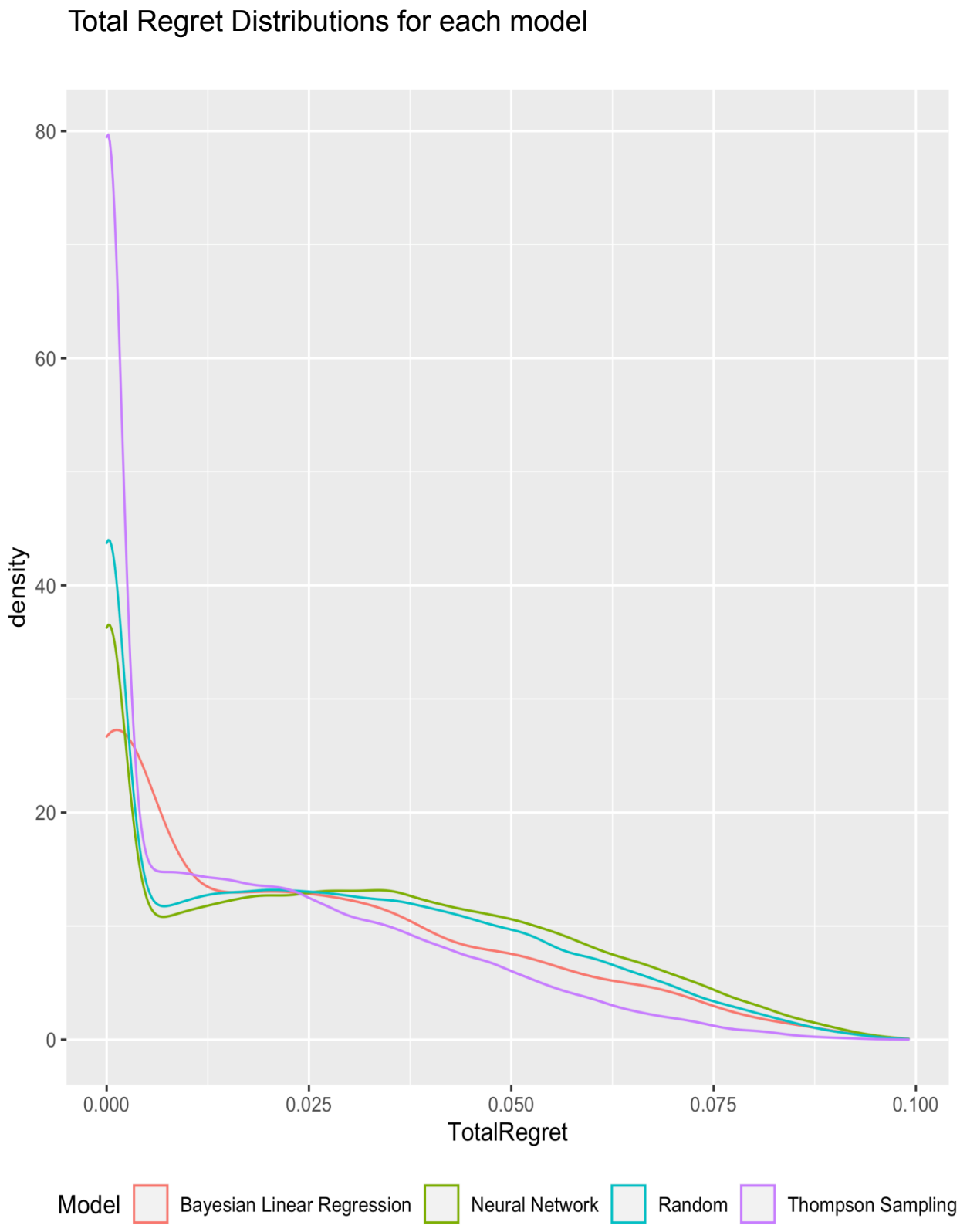

Figure 2: Total Regret Distributions for each model

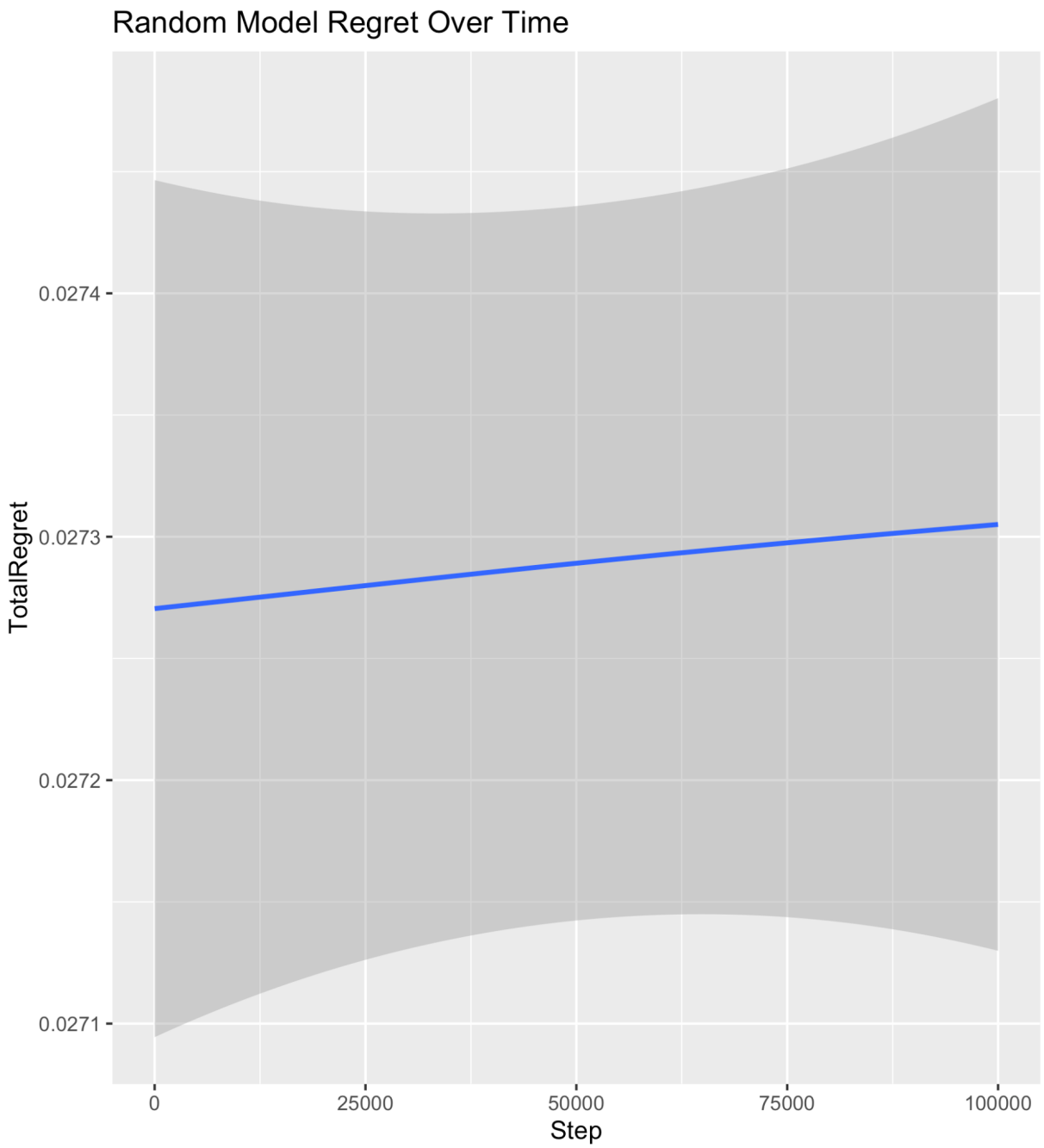

Figure 3: Random Model Regret over time

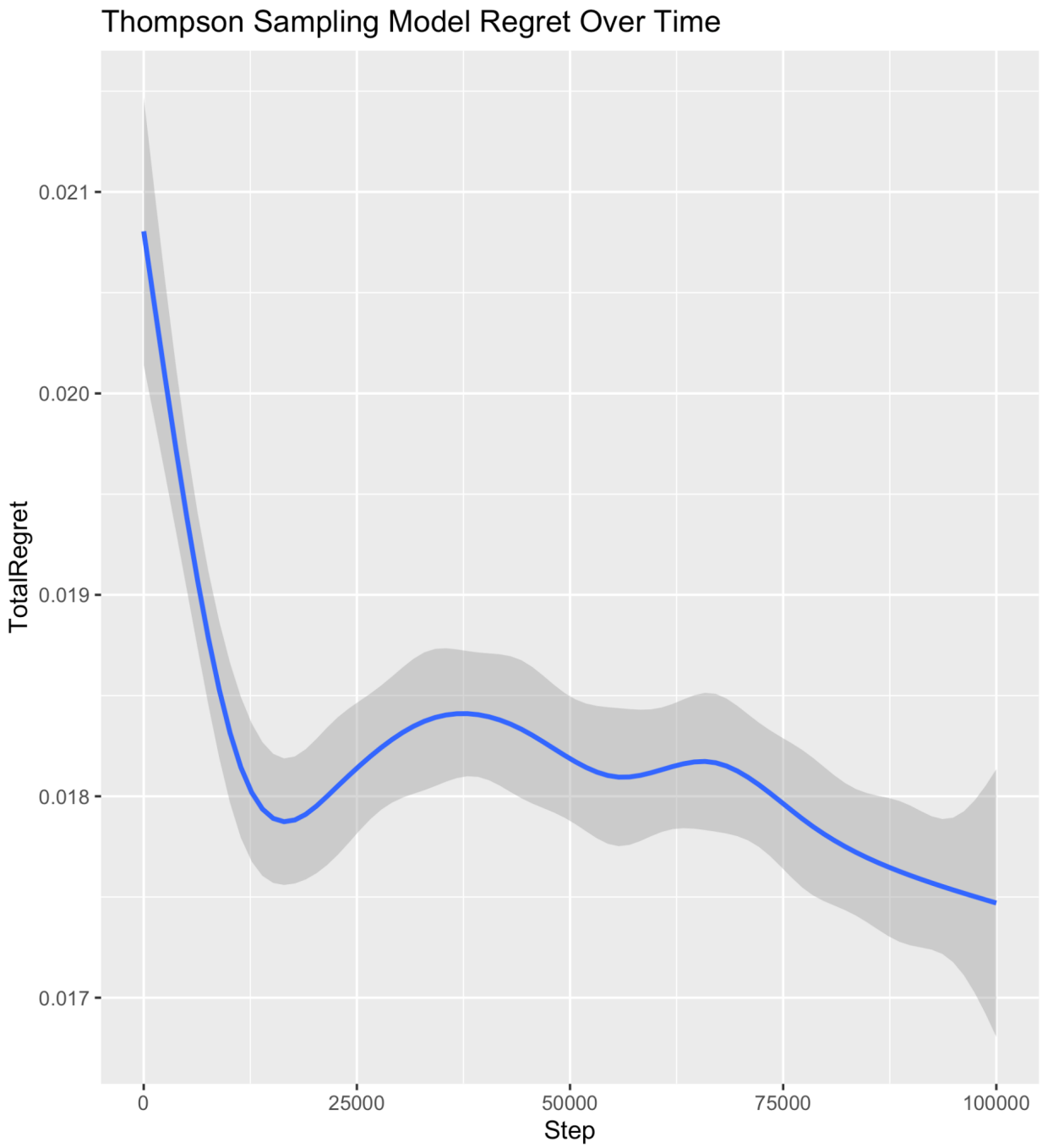

Figure 4: Thompson Sampling Model Regret over time

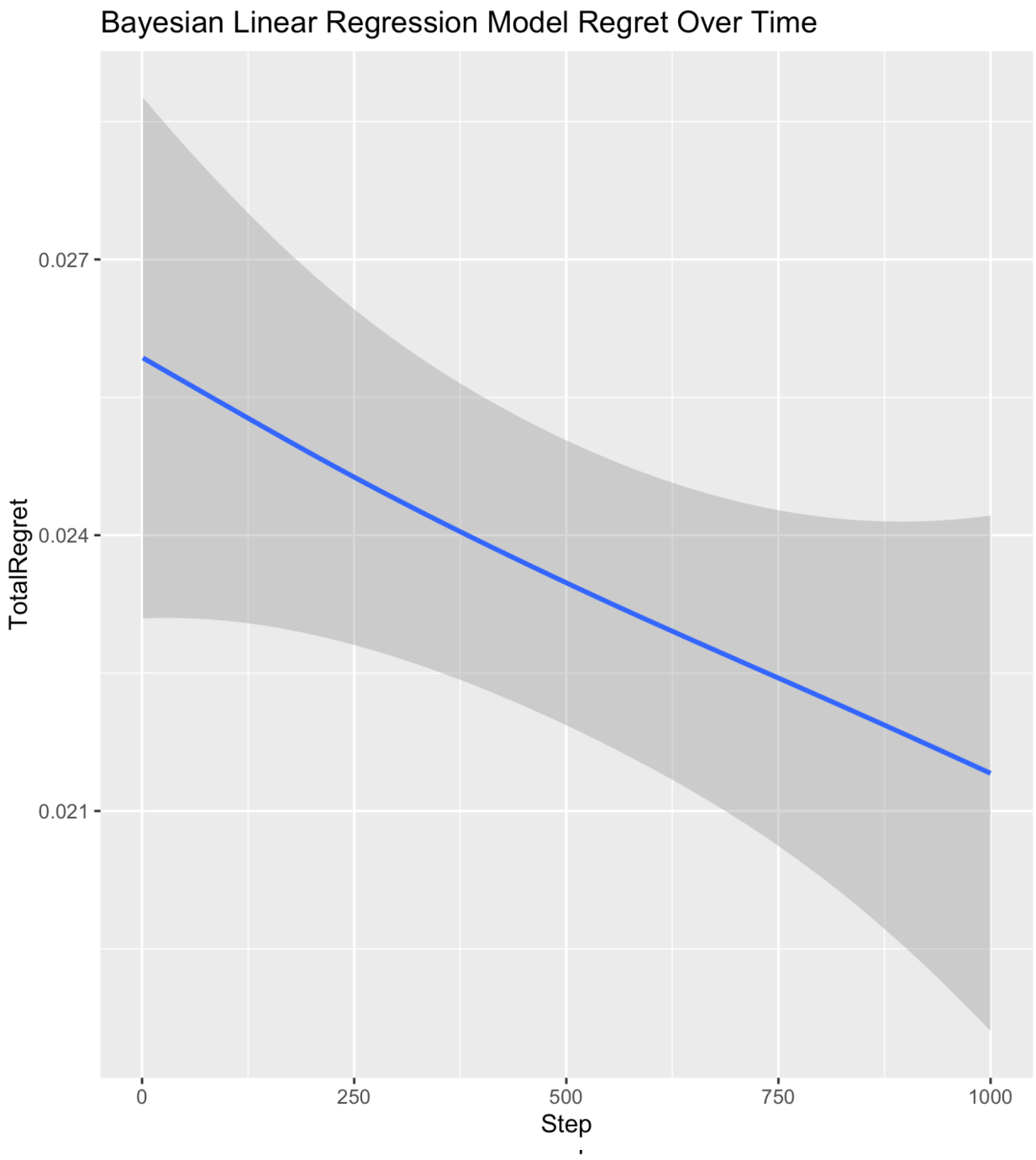

Figure 5: Bayesian Linear Regression Model Regret over time

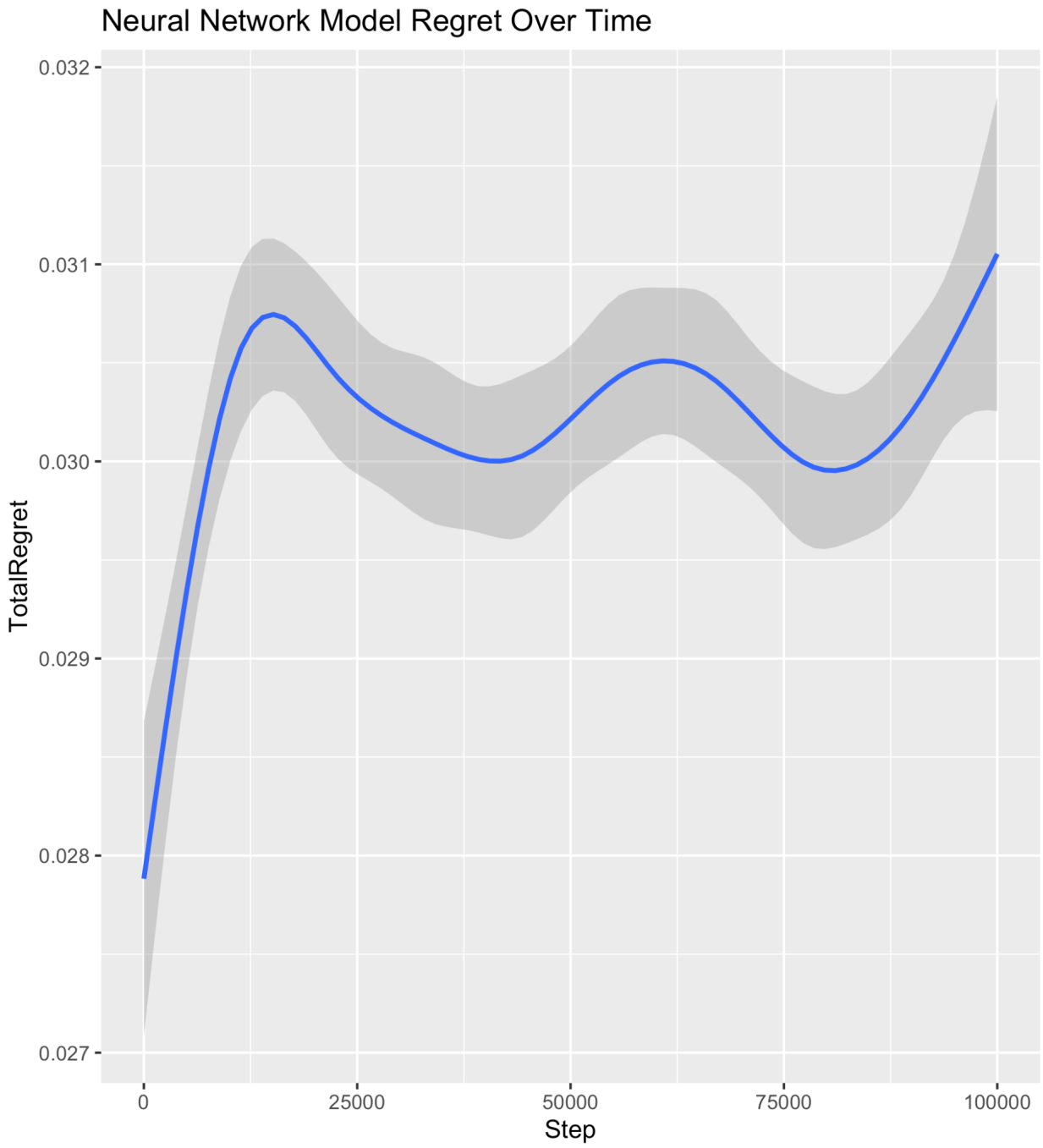

Figure 6: Neural Network Model Regret over time

Figures 3, 4, 5 ,6 are a significantly smoothened version of the data, as the raw data is quite noisy. But we can see in Figure 3 the random model performs the same over time, as we would expect. In Figure 4, Thompson Sampling model shows a downward trajectory as would be expected, but the trend slows down around 10K steps in. I would speculate that after this level the modifiers are injecting noise and causing issues for the model. The Bayesian linear regression, in Figure 5, has the best trajectory with a direct downwards trend, however, the slope is not quite as fast as the Thompson Sampling model. In Figure 6, the neural network model is surprisingly getting worse over time, which probably means that the learning rate on the model needs further tweaking or this could be a temporary blip and with more iterations this would clear up.

Hypothesis	T Statistic	Degrees of Freedom	P-Value	Result
$H_0: Regret_{Random} \leq Regret_{TS}$	-93.084	195061	<2.2e-16	Reject null
$H_0: Regret_{Random} \leq Regret_{BLR}$	-5.0812	1019.8	2.22e-07	Reject null
$H_0: Regret_{Random} \leq Regret_{NN}$	27.268	199861	1	Failed to reject null

Table1: Comparison of chosen models versus random model

The table above shows that both Bayesian linear regression and Thompson Sampling have statistically less total regret than a random model, but the neural network does not. Further analysis shows that compared to Bayesian linear regression, Thompson Sampling has less total regret.

Hypothesis	T Statistic	Degrees of Freedom	P-Value	Result
$H_0: Regret_{BLR} \leq Regret_{TS}$	-7.3202	1014.1	2.517e-13	Reject null

Table 2: Comparison between Bayesian Linear Regression and Thompson Sampling

Conclusion

This project showed that despite its simplicity Thompson Sampling is a powerful technique and if there isn't significant variation in the modifiers (day of week, week of year, holidays) then Thompson Sampling is dependable and works well most of the time.

Bayesian Linear Regression clearly shows promise with the significant downward trajectory, however, without significant variation in the modifiers, it is unable to leverage the context to provide a substantial improvement over Thompson Sampling.

The Neural Network model performance was surprising, but as stated previously, it is possible that given the reinforcement learning scenario, different tactics need to be tested to maximize its performance.

However, as Eduardo Gianetti stated “The advance of scientific knowledge brings more questions than answers,” [8]. Going forward there are many areas of further research that can be explored and variations of solutions to test.

Further Avenues to Explore

The three models that were tested are a small part of the universe of models that can be explored. Some area of further exploration could be:

- Exploration of other distributions for the final output in the neural network (instead of the Beta distribution) and the Bayesian linear regression (instead of the log-normal distribution), this can also be extended by creating a variation of Thompson Sampling, adapted to other distributions
- Having a multi epoch run in the neural network model instead of the current single step, this would enable faster learning in terms of steps taken and could also help reduce any catastrophic forgetting [9] that may be happening during the reinforcement learning
- Other probabilistic methods can also be explored such as Bayesian mixed effects models and Bayesian hierarchical models
- Both the Bayesian linear regression and neural network models can be modified to be a single model for all the actions rather than having one model per action. This will result in the models being able to capture interactions effects between actions and states, however will require potentially richer data
- Inclusion of other state information such as banner information (average color, color range, pixel size, etc.), website information (bounce rate, average time on site, etc.) can also potentially help determine effectiveness, especially if a single model is used to predict all the actions
- Currently, the environment does not account for banner burnout, which is when users stop responding to a banner ad that they have already seen multiple times, adding this would help simulate the reality of online advertising better

Implementation as a Product

The goal of this project was to create a proof-of-concept and determine if there was scope in optimizing digital advertising decisions. Of course, this is the first step in a long process but the following ideas may be useful:

- A preliminary process should be implemented to estimate the level of variation of traffic that can be expected for a given ad and website, this can be done by analyzing data provided by the website or by purchasing data from 3rd party providers. Based on this estimation of variation of traffic a more complex model than can be used e.g. Bayesian Linear Regression
- In a simple implementation of Thompson Sampling, historical data is not taken into account. This can be done in multiple ways, but one potential way to “warm start” the Beta distributions would be to run a standard linear regression on historical data and use that to rank the options and adjust the Beta distributions to provide slightly more advantage to the higher predicted options
- One potential issue with using Thompson Sampling can be that over time, as the environment changes, the model may take a long time to adjust. For example, let’s say an ad-website combination has been doing well for six months and the Thompson Sampling model has become quite confident (evidenced by the level of alpha parameter values versus the beta parameter values). Now, if the market changes, e.g. end-uses start burning out on the ad, it might take awhile for the Thompson Sampling model to try other options. One potential solution to this would be to limit the total amount that the parameters can reach e.g. alpha < 100. This will ensure that the model cannot be too confident at any point in time, though this may result in slightly reduced performance. The correct limit should be determined with testing
- A potential way to determine if/when an environment requires a more complex model than Thompson Sampling would be to run multiple models in the background and measure their performance based on real-world results. For example, let’s say we are currently using a Thompson Sampling model for a given user. In the background, we can also train the Bayesian Linear Regression model and predict the CTR for a given state and each option. When we get the CTR values we can compare them to the Bayesian Linear Regression prediction and see how well it is doing. If, over time, we see the Bayesian Linear Regression is doing well we can switch over and see if it performs better

In closing, this project was a significant learning experience for me. I learned about reinforcement learning, digital marketing, probabilistic programming and neural networks.

References

1. Sutton H. Peter Morgan Sutton. *BMJ*. 2014;348(mar31 11):g2466-g2466. doi:10.1136/bmj.g2466
2. Silver D, Huang A, Maddison CJ, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*. 2016;529(7587):484-489. doi:10.1038/nature1696
3. Markov property. Wikipedia. Published April 22, 2022. Accessed July 6, 2022. https://en.wikipedia.org/wiki/Markov_property
4. Wind J, Mahajan V. Digital Marketing. *Symphonya Emerging Issues in Management*. 2002;1(1). doi:10.4468/2002.1.04wind.mahajan
5. Gym Documentation. www.gymlibrary.ml. https://www.gymlibrary.ml/
6. Charnock T, Perreault-Levasseur L, Lanusse F. Bayesian Neural Networks. *arXiv:200601490 [astro-ph, stat]*. Published online November 6, 2020. Accessed July 10, 2022. https://arxiv.org/abs/2006.01490
7. *Pattern Recognition and Machine Learning*. https://link.springer.com/book/9780387310732(probabilistic programming)
8. Giannetti E, Gledson J. *Lies We Live by : The Art of Self-Seduction*. Bloomsbury; 2000.
9. Kirkpatrick J, Pascanu R, Rabinowitz N, et al. Overcoming catastrophic forgetting in neural networks. arXiv.org. Published 2016. https://arxiv.org/abs/1612.00796
10. Johnson J. Global Digital Population 2021. Statista. Published April 26, 2022. https://www.statista.com/statistics/617136/digital-population-worldwide/
11. Kiani GR. Marketing Opportunities in the Digital World. *Internet Research*. 1998;8(2):185-194. Accessed October 8, 2021. https://eric.ed.gov/?id=EJ566628
12. Melo L, Martins L, Oliveira B, Brandão B, Soares D, Lima T. PulseRL: Enabling Offline Reinforcement Learning for Digital Marketing Systems via Conservative Q-Learning Deep Learning Brazil Deep Learning Brazil Deep Learning Brazil Deep Learning Brazil. Accessed June 15, 2022. https://offline-rl-neurips.github.io/2021/pdf/9.pdf
13. Chatterjee P, Hoffman DL, Novak TP. Modeling the Clickstream: Implications for Web-Based Advertising Efforts. *Marketing Science*. 2003;22(4):520-541. doi:10.1287/mksc.22.4.520.24906
14. Richardson M, Dominowska E, Ragno R. Predicting clicks. *Proceedings of the 16th international conference on World Wide Web - WWW '07*. Published online 2007. doi:10.1145/1242572.1242643
15. Zhou G, Zhu X, Song C, et al. Deep Interest Network for Click-Through Rate Prediction. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Published online July 19, 2018. doi:10.1145/3219819.3219823

16. Saura JR, Palos-Sánchez P, Cerdá Suárez LM. Understanding the Digital Marketing Environment with KPIs and Web Analytics. *Future Internet*. 2017;9(4):76. doi:10.3390/fi9040076
17. Lakshmanarao A, Ushanag S, Leela BS. Ad Prediction using Click Through Rate and Machine Learning with Reinforcement Learning. IEEE Xplore. doi:10.1109/ICECCT52121.2021.9616653
18. Cai H, Ren K, Zhang W, et al. Real-Time Bidding by Reinforcement Learning in Display Advertising. *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. Published online February 2, 2017. doi:10.1145/3018661.3018702
19. Jin J, Song C, Li H, Gai K, Wang J, Zhang W. Real-Time Bidding with Multi-Agent Reinforcement Learning in Display Advertising. *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. Published online October 17, 2018. doi:10.1145/3269206.3272021
20. Wu D, Chen X, Yang X, et al. Budget Constrained Bidding by Model-free Reinforcement Learning in Display Advertising. *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. Published online October 17, 2018. doi:10.1145/3269206.3271748
21. Zhao J, Qiu G, Guan Z, Zhao W, He X. Deep Reinforcement Learning for Sponsored Search Real-time Bidding. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Published online July 19, 2018. doi:10.1145/3219819.3219918
22. Liu M, Li J, Hu Z, Liu J, Nie X. A Dynamic Bidding Strategy Based on Modelfree Reinforcement Learning in Display Advertising. *IEEE Access*. Published online 2020:1-1. doi:10.1109/access.2020.3037940
23. Collier M, Llorens HU. Deep Contextual Multi-armed Bandits. *arXiv:180709809 [cs, stat]*. Published online July 25, 2018. Accessed July 18, 2022. https://arxiv.org/abs/1807.09809
24. Baccot B, Grigoras R, Charvillat V. Reinforcement Learning for Online Optimization of Banner Format and Delivery. Online Multimedia Advertising: Techniques and Technologies. Published 2011. Accessed July 18, 2022. https://www.igi-global.com/chapter/online-multimedia-advertising/51952

Appendix

Main.py

```
import os
from datetime import datetime

import numpy as np
from tqdm import tqdm

from policy import Policy
from environment import DigitalAdvertisingEnv

output_dir = './output'
now_str = str(datetime.now()).replace(':', '-').replace(' ', '_').split('.')[0]
if not os.path.exists(output_dir):
   os.makedirs(output_dir)
creatives = [(1,2), (0.5,0.5), (2,1)]
websites = [(1,1), (3,2), (3,4)]
modifiers = {'mon' : (0.5,1), 'tue' : (1,1), 'wed' : (2,2),
   'thu' : (2,2), 'fri' : (1,10), 'sat' : (3,2), 'sun' : (3,2),
   'w50' : (5,3), 'holiday' : (3,2)}

env = DigitalAdvertisingEnv(creatives, websites, modifiers)
obs = env.reset()
random_policy = Policy(
   model_type = 'random',
   n_creatives = len(creatives),
   n_websites = len(websites),
   n_features = len(obs),
)
ts_policy = Policy(
   model_type = 'ts',
   n_creatives = len(creatives),
   n_websites = len(websites),
   n_features = len(obs),
)
blr_policy = Policy(
   model_type = 'blr',
   n_creatives = len(creatives),
   n_websites = len(websites),
   n_features = len(obs),
)
nn_policy = Policy(
   model_type = 'nn',
   n_creatives = len(creatives),
   n_websites = len(websites),
   n_features = len(obs),
)
```

```
n_trials = 100_000

# Bayesian Linear Regression Policy
print("--- Bayesian Linear Regression Policy ---")
blr_trials = 5000
blr_regret = []
for i in tqdm(range(blr_trials)):
   action = blr_policy.predict(obs)
   new_obs, reward, done, info = env.step(action)
   blr_regret.append(info['total_regret'])
   blr_policy.update(obs, action, reward)
   obs = new_obs

print(f"Cumulative Regret: {np.sum(blr_regret)}")
print(f"Mean Regret: {np.mean(blr_regret)}")
print(f"Std Dev Regret: {np.std(blr_regret)}")
with open(os.path.join(output_dir, f'{now_str}_blr.csv'), 'w') as f:
   for i in range(blr_trials):
       f.write(f"{blr_regret[i]}\n")

# Neural Network Policy
print("--- Neural Network Policy ---")
nn_regret = []
for i in tqdm(range(n_trials)):
   action = nn_policy.predict(obs)
   new_obs, reward, done, info = env.step(action)
   nn_regret.append(info['total_regret'])
   nn_policy.update(obs, action, reward)
   obs = new_obs

print(f"Cumulative Regret: {np.sum(nn_regret)}")
print(f"Mean Regret: {np.mean(nn_regret)}")
print(f"Std Dev Regret: {np.std(nn_regret)}")
with open(os.path.join(output_dir, f'{now_str}_nn.csv'), 'w') as f:
   for i in range(n_trials):
       f.write(f"{nn_regret[i]}\n")

# Random Policy
print("--- Random Policy ---")
random_regret = []
for i in tqdm(range(n_trials)):
   action = random_policy.predict(obs)
   new_obs, reward, done, info = env.step(action)
   random_regret.append(info['total_regret'])
   random_policy.update(obs, action, reward)
   obs = new_obs

print(f"Cumulative Regret: {np.sum(random_regret)}")
print(f"Mean Regret: {np.mean(random_regret)}")
print(f"Std Dev Regret: {np.std(random_regret)}")
with open(os.path.join(output_dir, f'{now_str}_random.csv'), 'w') as f:
   for i in range(n_trials):
```

```
        f.write(f"{random_regret[i]}\n")

# Thompson Sampling Policy
print("--- Thompson Sampling Policy ---")
ts_regret = []
for i in tqdm(range(n_trials)):
    action = ts_policy.predict(obs)
    new_obs, reward, done, info = env.step(action)
    ts_regret.append(info['total_regret'])
    ts_policy.update(obs, action, reward)
    obs = new_obs

print(f"Cumulative Regret: {np.sum(ts_regret)}")
print(f"Mean Regret: {np.mean(ts_regret)}")
print(f"Std Dev Regret: {np.std(ts_regret)}")
with open(os.path.join(output_dir, f'{now_str}_ts.csv'), 'w') as f:
    for i in range(n_trials):
        f.write(f"{ts_regret[i]}\n")
```

Environment.py

```
from datetime import date, timedelta
import holidays
import numpy as np
from gym import Env, spaces

class DigitalAdvertisingEnv(Env):
   """Class for representing the digital advertising environment
   """
   def __init__(self, creatives, websites, modifiers,
       base = (1,1), start_date = None):
       """Initializes the digital advertising environment

       :param creatives: distribution parameters for the creatives
       :type creatives: tuple/list with the values as a tuple of the Beta
           distribution parameters, e.g. [(1,1.5), (2,2)]
       :param websites: distribution parameters for the websites
       :type websites: tuple/list with the values as a tuple of the Beta
           distribution parameters, e.g. [(1,1.5), (2,2)]
       :param modifiers: distribution parameters for the modifiers, currently
           the following modifiers are supported: day of week (`mon`, `tue`, etc.),
           week of year (`w0`, `w1`, etc.), holidays (`holiday`)
       :type modifiers: dict with the keys as the modifier and the value
           as a tuple e.g. {'mon' : (1,2), 'holiday' : (3,1), 'w50' : (5,1)}
       :param base: the base Beta distribution parameters to sample the CTR,
           defaults to (1,1)
       :type base: tuple of the form (<alpha>, <beta>), values are the
           distribution parameters e.g. (1,1)
       :param start_date: the start date for simulation,
           defaults to date(1990,1,1)
       :type start_date: datetime.date object, optional
       """
       super(DigitalAdvertisingEnv, self).__init__()
       # Parameter assertions
       assert all([y > 0 for x in creatives for y in x]), \
           f"Creative parameters not valid: {creatives}"
       assert all([y > 0 for x in websites for y in x]), \
           f"Websites parameters not valid: {websites}"
       assert all([y > 0 for x in modifiers.values() for y in x]), \
           f"Modifiers parameters not valid: {modifiers}"
       assert all([x > 0 for x in base]), f"Base parameters not valid: {base}"
       assert start_date is None or isinstance(start_date, date), \
           f"Invalid date: {start_date}"
       # Setup variables
       self.creatives = creatives
       self.websites = websites
       self.modifiers = modifiers
       self.base = base
       self.start_date = date(1990,1,1) if start_date is None else start_date
       # Setup gym variables
```

```
        self.observation_space = spaces.MultiDiscrete([2, 7, 53])
        self.action_space = spaces.MultiDiscrete([len(creatives), len(websites)])

    def _is_holiday(self):
        return int(self.today in
            holidays.UnitedStates(years = self.today.year).keys())

    def _get_day_of_week(self):
        return self.today.weekday()

    def _get_week_of_year(self):
        return self.today.isocalendar()[1] - 1

    def _get_obs(self):
        self.today = self.today + timedelta(days = 1)
        output = (
            self._is_holiday(),
            self._get_day_of_week(),
            self._get_week_of_year(),
        )
        return output

    def _get_modifier_params(self):
        params = []
        # Day modifier
        days = ['mon', 'tue', 'wed', 'thu', 'fri', 'sat', 'sun']
        day_of_week = days[self._get_day_of_week()]
        if day_of_week in self.modifiers:
            params.append(self.modifiers[day_of_week])
        # Week modifier
        week_of_year = f"w{self._get_week_of_year()}"
        if week_of_year in self.modifiers:
            params.append(self.modifiers[week_of_year])
        # Holiday modifier
        if self._is_holiday() == 1 and 'holiday' in self.modifiers:
            params.append(self.modifiers['holiday'])
        return params

    def _get_action_params(self, action):
        return [self.creatives[action[0]], self.websites[action[1]]]

    def _calc_ctr(self, alpha, beta):
        return np.random.beta(alpha, beta) / 10

    def reset(self):
        self.today = self.start_date
        return self._get_obs()

    def step(self, action):
        # Get inputs
        modifiers = self._get_modifier_params()
        action_params = self._get_action_params(action)
```

```
        params = [*modifiers, *action_params]
        # Calculate Beta params
        alpha = np.mean([x[0] for x in params])
        beta = np.mean([x[1] for x in params])
        # Sample distribution
        reward = self._calc_ctr(alpha, beta)
        obs = self._get_obs()
        # Calculate total regret
        combos = [(x,y) for x in range(len(self.creatives))
            for y in range(len(self.websites))
            if x != action[0] and y != action[1]]
        best_reward = reward
        best_action = action
        for combo in combos:
            combo_params = self._get_action_params(combo)
            params = [*modifiers, *combo_params]
            alpha = np.mean([x[0] for x in params])
            beta = np.mean([x[1] for x in params])
            combo_reward = self._calc_ctr(alpha, beta)
            if combo_reward > best_reward:
                best_reward = combo_reward
                best_action = combo
        info = {
            'total_regret' : best_reward - reward,
            'best_action' : best_action
            }
        return obs, reward, False, info
```

Policy.py

```
import pandas as pd
import numpy as np
import pymc3 as pm
import torch
import torch.nn as nn
import torch.optim as optim
from torch.distributions import Beta

class Policy():
   """Generalized class representing a policy
   """
   def __init__(self, model_type, n_creatives, n_websites, n_features):
       """Initialization function for policy object

       :param model_type: model type, one of 'ts', 'blr', 'nn', 'random'
       :type model_type: str
       :param n_creatives: the number of creatives being considered
       :type n_creatives: int
       :param n_websites: the number of websites being considered
       :type n_websites: int
       :param n_features: the number of features in the observation
       :type n_features: int
       """
       super(Policy, self).__init__()
       valid_models = ['ts', 'blr', 'nn', 'random']
       assert model_type in valid_models, \
           f"Incorrect model input: {model_type}"
       self.model_type = model_type
       if self.model_type == 'random':
           self.model = RandomModel(n_creatives,
               n_websites, n_features)
       elif self.model_type == 'ts':
           self.model = ThompsonSamplingModel(n_creatives,
               n_websites, n_features)
       elif self.model_type == 'blr':
           self.model = BayesianLinearRegressionModel(n_creatives,
               n_websites, n_features)
       elif self.model_type == 'nn':
           self.model = NeuralNetworkModel(n_creatives,
               n_websites, n_features)

   def predict(self, obs):
       return self.model.predict(obs)

   def update(self, obs, action, reward):
       self.model.update(obs, action, reward)


class RandomModel():
   """Random model, returns a random action, used for baseline predictions
```

```
    """
    def __init__(self, n_creatives, n_websites, n_features):
        super(RandomModel, self).__init__()
        self.n_creatives = n_creatives
        self.n_websites = n_websites

    def predict(self, obs):
        return (np.random.randint(self.n_creatives),
            np.random.randint(self.n_websites))

    def update(self, obs, action, reward):
        return


class ThompsonSamplingModel():
    """Thompson sampling algorithm for continuous rewards"""
    def __init__(self, n_creatives, n_websites, n_features, alpha = 2, beta = 1, n_samples =
10,
        baseline = 0):
        """Initialization function for Thompson Sampling Model

        :param n_creatives: number of creatives
        :type n_creatives: int
        :param n_websites: number of websites
        :type n_websites: int
        :param alpha: starting alpha parameter, defaults to 2
        :type alpha: int, optional
        :param beta: starting beta parameter, defaults to 1
        :type beta: int, optional
        :param n_samples: number of samples to take when choosing action, defaults to 10
        :type n_samples: int, optional
        :param baseline: baseline value for maximum reward, defaults to 0
        :type baseline: int, optional
        """
        super(ThompsonSamplingModel, self).__init__()
        self.n_creatives = n_creatives
        self.n_websites = n_websites
        self.action_list = [(x,y) for x in range(n_creatives) for y in range(n_websites)]
        self.n_actions = len(self.action_list)
        self.alpha = alpha
        self.beta = beta
        self.n_samples = n_samples
        self.shape_params = [[alpha, beta] for _ in range(self.n_actions)]
        self.max_reward = baseline
        self.min_reward = np.inf

    def predict(self, obs):
        reward_samples = [ # Shape: (n_actions, n_samples)
            [np.random.beta(*self.shape_params[j])
                for _ in range(self.n_samples)]
            for j in range(self.n_actions)]
        action = np.argmax(np.mean(reward_samples, axis = 1))
```

```
        return self.action_list[action]

    def update(self, obs, raw_action, reward):
        action = self.action_list.index(raw_action)
        if reward > self.max_reward:
            self.max_reward = reward
        if reward < self.min_reward:
            self.min_reward = reward
        # Scale reward between 0 and 1
        if self.min_reward != self.max_reward:
            scaled_reward = (reward - self.min_reward) / (self.max_reward - self.min_reward)
        else:
            scaled_reward = 0.5
        # Sample from binomial if 0 then beta++ if 1 then alpha++
        sampled_b = np.random.binomial(1, scaled_reward)
        # Update alpha or beta of distribution
        self.shape_params[action][1 - sampled_b] += 1


class ReplayMemory(object):
    def __init__(self, columns):
        self.columns = columns
        self.memory = {c : [] for c in columns}

    def push(self, datum):
        """Save a transition"""
        assert all([c in datum for c in self.columns])
        for col in self.columns:
            self.memory[col].append(datum[col])

    def __len__(self):
        return len(pd.DataFrame(self.memory))

    def get_data(self):
        return pd.DataFrame(self.memory)


class BayesianLinearRegressionModel():
    """Bayesian Linear Regression model
    """
    def __init__(self, n_creatives, n_websites, n_features):
        self.n_creatives = n_creatives
        self.n_websites = n_websites
        self.action_list = [(x,y) for x in range(n_creatives)
            for y in range(n_websites)]
        self.models = [BayesianLinearRegression(n_features)
            for _ in range(len(self.action_list))]
        self.memories = [
            ReplayMemory(
                ['is_holiday', 'day_of_week', 'week_of_year', 'ctr'])
            for _ in range(len(self.action_list))]
```

```
    def predict(self, obs):
        if any([x.model is None for x in self.models]):
            best_action = np.random.choice(
                [i for i in range(len(self.models))
                if self.models[i].model is None])
        else:
            best_action = np.argmax(
                [
                    m.predict(pd.DataFrame(
                            {
                                'is_holiday' : [obs[0]],
                                'day_of_week' : [obs[1]],
                                'week_of_year' : [obs[2]],
                            }))
                        for m in self.models
                ]
            )
        return self.action_list[best_action]

    def update(self, obs, action, reward):
        idx = self.action_list.index(action)
        self.memories[idx].push({'is_holiday' : obs[0],
            'day_of_week' : obs[1], 'week_of_year' : obs[2],
            'ctr' : reward})
        self.models[idx].update(self.memories[idx].get_data())


class BayesianLinearRegression():
    """Bayesian linear regression algorithm"""
    def __init__(self, n_features):
        super(BayesianLinearRegression, self).__init__()
        self.n_features = n_features
        self.model = None
        self.trace = None

    def predict(self, datum):
        with self.model:
            pm.set_data({'x': datum})
            posterior_predictive = pm.sample_posterior_predictive(self.trace)
        return posterior_predictive['ctr'].mean()

    def update(self, train_data):
        with pm.Model() as model:
            # Define data
            x = pm.Data('x', train_data[[x for x in train_data.columns if x != 'ctr']])
            y = pm.Data('y', train_data['ctr'])
            # Define priors
            sigma = pm.HalfCauchy('sigma', beta=10)
            intercept = pm.Normal('intercept', 0, 20)
            coeff = pm.Normal('coeff', 0, 20, shape = 3)

            # Define likelihood
```

```
            ctr = pm.LogNormal('ctr', mu = intercept + pm.math.dot(x, coeff),
                sigma = sigma, observed = y)
            trace = pm.sample(return_inferencedata = True, draws = 100,
                tune = 100, cores = 1)
        self.model = model
        self.trace = trace


class NeuralNetworkModel():
    """Neural Network model, with probabilistic outputs
    """
    def __init__(self, n_creatives, n_websites, n_features):
        self.n_creatives = n_creatives
        self.n_websites = n_websites
        self.action_list = [(x,y) for x in range(n_creatives)
            for y in range(n_websites)]
        self.n_features = self._format_input([0] * n_features).shape[0]
        self.models = [NeuralNet(self.n_features, [8]).float()
            for _ in range(len(self.action_list))]
        self.optimizers = [optim.AdamW(self.models[i].parameters())
            for i in range(len(self.models))]

    def _format_input(self, obs):
        is_holiday = nn.functional.one_hot(torch.tensor(obs[0]),
            num_classes = 2)
        day_of_week = nn.functional.one_hot(torch.tensor(obs[1]),
            num_classes = 7)
        week_of_year = nn.functional.one_hot(torch.tensor(obs[2]),
            num_classes = 53)
        formatted_input = torch.cat(
            [is_holiday, day_of_week, week_of_year])
        return formatted_input.float()

    def predict(self, obs):
        self.action_predictions = [m(self._format_input(obs)).sample()
            for m in self.models]
        return self.action_list[np.argmax(self.action_predictions)]

    def update(self, obs, action, reward):
        idx = self.action_list.index(action)
        self.optimizers[idx].zero_grad()
        loss = -self.models[idx](self._format_input(obs)) \
            .log_prob(self.action_predictions[idx]) * reward
        loss.backward()
        self.optimizers[idx].step()


class NeuralNet(nn.Module):
    """Neural network that predicts using a LogNormal distribution
    Parameters:
    - n_features (int): count of the number of features in the input
    - hidden_layers (list of int) <OPTIONAL>: a list of the number of nodes in each
```

```
        hidden layer
    """
    def __init__(self, n_features, hidden_layers = None):
        super(NeuralNet, self).__init__()
        hidden_layers = [] if hidden_layers is None else hidden_layers
        self.layers = nn.ModuleList()
        input_size = n_features
        for nodes in hidden_layers:
            self.layers.append(nn.Linear(input_size, nodes))
            input_size = nodes
        self.output = nn.Linear(input_size, 2)
        self.dist = None

    def forward(self, x):
        for l in self.layers:
            x = torch.relu(l(x))
        params = self.output(x)
        return Beta(
            torch.abs(params[0]) + 0.01,
            torch.abs(params[1]) + 0.01,
        )
```