LEARNING WITH ABANDONMENT

COEN 281 -Data Mining and Pattern Recognition

Anjaly George
Divya Arora

Acknowledgements

We would like to express our deep gratitude to Professor Ming Hwa Wang , our research supervisor, for his patient guidance, enthusiastic encouragement and useful critiques of this research work. We would also like to thank Mr. Ramesh Johari and Mr. Sven Schmit , the authors of the paper on which our research is based upon. We would also like to extend our thanks to the department of computer science and Santa Clara University. Finally, We wish to thank our parents for their support and encouragement throughout our study.

1.Table of Contents

1.1 Abstract

Consider a platform that wants to learn a personalized policy for each user, but the platform faces the risk of a user abandoning the platform if she is dissatisfied with the actions of the platform. For example, a platform is interested in personalizing the number of newsletters it sends, but faces the risk that the user unsubscribes forever. We propose a general thresholded learning model for scenarios like this, and discuss the structure of optimal policies. We describe salient features of optimal personalization algorithms and how feedback the platform receives impacts the results. Furthermore, we investigate how the platform can efficiently learn heterogeneity across users by interacting with a population and provide performance guarantees.

We are implementing 3 multi armed bandit strategies and they are Epsilon greedy,Thompson Sampling and Bayesian UCB and comparing the performances among these strategies by calculating their mean regrets.

2. Introduction

2.1 Objective

The objective is to generate a threshold learning model for platforms that want to learn a personalized policy for each user.Also , to investigate how the platform can efficiently learn the heterogeneity across users by interacting with a population and provide performance guarantees.

2.2 Problem- Learning With Abandonment

If a platform wants to have a personalized policy for the user but the platform faces the risk of the user abandoning the platform if dissatisfied with the actions of the platform. For example, a platform is interested in personalizing the number of newsletters it sends, but faces the risk that the user unsubscribes forever. General threshold models are implemented for scenarios like this.

2.3 Reasons why the project is related to Data Mining class

The goal of data mining is the extraction of patterns and knowledge from large amounts of data .This information can be used to increase revenues, cut costs, improve customer relationships, reduce risks and more. In the project Learning with Abandonment we are creating three models where different properties of datasets are

analyzed and with the help of the result the solution is made better .It fetches the users data and in order to create personalised policy for the user creates personalised policy generate threshold learning model.For example in one of the models with the help of user feedback the model is being continuously improved.

Also the paper was presented in ICDM 2018 in Data Mining conference.

2.4 Why other approach is not good

In the paper there is implementation of 3 strategies but we are exploring more strategies.

We are planning to implement different strategies for Learning Threshold model .We will be implementing the following:-

- Bayesian UCB
- Thompson Sampling
- ε-Greedy Algorithm

In the UCB algorithm, we do not assume any prior on the reward distribution and therefore we have to rely on Hoeffding's Inequality for a very generalized estimation. In Bayesian UCB able to know the distribution upfront, we would be able to make better bound estimation.Thompson is more optimized for maximizing long-term overall payoff.

2.5 Why we think our approach is better

The paper is restricted to fewer approaches for the Learning Threshold model .We are exploring more strategies and we will be comparing which strategy is better .

In the UCB algorithm, we do not assume any prior on the reward distribution and therefore we have to rely on Hoeffding's Inequality for a very generalized estimation. In Bayesian UCB able to know the distribution upfront, we would be able to make better bound estimation.

Epsilon Greedy algorithm is the greediest of all the multi armed bandit algorithms. We keep the value of epsilon between 0 and 1. The higher the value of epsilon the algorithm promotes more exploration. We aim to follow this strategy in the beginning to identify the set of actions based on the users thresholds and later on reduce the value of the epsilon to reduce the risk of abandonment and we aim to identify different structures in the personalized policies.

The Thompson sampling is a more principled approach and helps us to yield and identify more optimal results in marginal cases. We sample one possible success rate from the beta distribution that corresponds to each variant for each new contact, and assign the contact to the variant with the largest sampled success rate. The more data points we've observed, the more confident we'll be about the true success rate, and as we collect more data, the sampled success rates will be increasingly close to the real

rate.So our aim is to explore different strategies to compare the differences among them.

2.6 Area or scope of investigation

There are directions of further research:-

Abandonment models- users playing a learning strategy herself, comparing this platform to one or multiple outside options. In this scenario, the user and platform are simultaneously learning about each other.

User information- User activity seems like an important signal of her preferences. Models that are able to incorporate such information and are able to infer the parameters from data are beyond the scope of this work but an important direction of further research

Empirical analysis- This work focuses on theoretical understanding of the abandonment model, and thus ignores important aspects of a real world system. We believe there is a lot of potential to gain additional insight from an empirical perspective using real-world systems with abandonment risk.

3.Theoretical Bases And Literature Review

3.1 Definition Of The Problem

Machine learning algorithms are present in almost every area, humans are communicating widely with machines either knowingly or unknowingly. It is important for these algorithms to learn continuously and optimize themselves to provide a better customer experience.

The problem we are solving is the risk of abandonment of a platform by the user, if the platform is acting against the users preference. Therefore the algorithms designed by the platform needs to ensure that they are not losing any of its customers.

The risk of losing customers is faced by many platform like online content creators like news agencies, they could lose their customers by sending way too many mails, or a platform that is optimizing the energy consumption can become so stringent in its policies so the customer might abandon the platform, so it is very important for a platform to optimize its learning strategies.

We concentrate in this research on gaining insight into the structure of optimal learning approaches in these environments. We are particularly interested in understanding

when such strategies take on a “simple” structure and focus on that throughout the project.

3.2 Related Research To Solve The Problem

The learning with abandonment problem is quite unique, and we are aware of only one prominent work that addresses the similar scenario. Apart from this work, Lu et al. 2017 also models the customer abandonment problem using two actions which are classified as the safe action and the risky action. This difference leads to different kinds of results as that of our work.

Our problem appears to be related to many reinforcement learning problems due to the dynamic structure of both the problems. However it is to be noted that there are many significant differences.

Another related work is on safe reinforcement learning, where we need to stay clear of catastrophic states, unlike the safe reinforcement learning, in our problem avoiding abandonment is not set as a hard constraint.

3.3 Advantage/Disadvantage Of Those Research

Those researches in the paper for finding out a set of actions that minimizes the risk of users abandoning the platform are mostly focusing on UCB, MOSS index of the multi armed bandit strategy.

The advantages of those researches is that they are able to observe how constant policies are optimal under fixed threshold and independent threshold models, also they have shown that under small perturbations constant policies are more optimal.

The Disadvantages of the research is that the research is only focusing on UCB, MOSS and explore exploit strategy. There are more multi armed bandit strategies that need to be explored further and we are focusing our research on Bayesian UCB, Thompson sampling and Epsilon Greedy sampling.
Also,additional user information in terms of covariates is not considered in the paper For example, in the notification example, user activity seems like an important signal of her preferences

3.4 Our Solution To Solve This Problem

We are approaching the problem with three different models which helps us to develop insight into the structure of optimal learning strategies in setting up personalized policies. We are particularly interested in understanding when such strategies take on a “simple” structure.
The models used are:

3.4.1 Threshold model

In this model we consider a platform that interacts with only a single user over time. The user has a threshold θ drawn from a distribution F, and at each time t = 0, 1, 2, . . . At each time t ,the platform chooses an action xt . If the chosen action xt ever exceeds the

threshold θ, the user abandons the platform; otherwise, the user continues to use the platform , and the platform earns some reward dependent on the xt chosen by the platform.

In this model we consider the case where the distribution F and the reward function are known, and the challenge is finding an optimal strategy for a given new user. For finding the optimal strategy we consider the problem of maximizing the expected discounted reward by the platform. We might expect that the optimal policy is increasing and depends on the discount factor: also, we might try to serve the user at increasing levels of input xt as long as the user does not abandon the platform .However, the main result shows this is not exactly the case, the static policy of maximizing one-step reward is optimal for this problem. It is because the user abandons if the threshold is ever crossed, and then trying to actively learn the threshold is not helpful anymore.

3.4.2 Learning thresholds

In this model we consider the problem of how to adapt the results when F and/or the reward function are unknown. In this case the model tries to learn the threshold over multiple user arrivals.

We concentrate our attention on the fixed threshold model and consider a setting where n users arrive sequentially, each with a fixed threshold u (u = 1, ... ,n) from the unknown distribution F with support on [0,1]. In order to highlight the importance of learning from

users over time , we find a stylised environment where the platform communicates with one user at a time, agreeing on all activities and monitoring the effects for that user before the next user arrives. We consider a proposed algorithm that uses a policy that is constant to each user. Additionally, we assume that the Rt(x) incentives are bounded between 0 and 1, but otherwise derived from an arbitrary distribution depending on x.
We test the performance of learning algorithms against the oracle that has total knowledge of the threshold distribution F and the reward function r, but no access to random variables realizations.

3.4.3 Feedback

The third model is a more general model with “soft” abandonment. In this case the user might not abandon the platform after a negative experience, but continue with the platform with some probability. We characterize the structure of an optimal policy to maximize expected discounted reward on a per-user basis; in particular, we find that the policy adaptively experiments until it has sufficient confidence, and then commits to a static action. We empirically investigate the structure of the optimal policy as well.

We are extending the concept as follows to integrate customer input. Assume that if the current action xt reaches the threshold (i.e., xt > t), then with probability p we obtain no reward but that the user persists, and with probability 1 − p the user leaves. As before the aim is to optimize discounted reward anticipated. Even when the threshold is reached the platform does not obtain a reward, the question is non trivial even though p

= 1. We limit our focus to the single threshold model, where a single one is drawn and then set for all periods of time.

3.5 Where our Solution Different From Others

As most of the related studies are way different from our research area it is difficult to exactly map the advantages and disadvantages of those works from ours.

As the Lu et al. 2017 models the customer abandonment problem using two actions which are classified as the safe action and the risky action. This difference leads to different kinds of results as that of our work.

Our model has many differences as that of reinforcement learning. The main focus of our work is developing personalized experience for each user, which when viewed through the perspective of reinforcement learning corresponds to having only single episodes or users to learn, which is independent of other users or episodes. On the other hand, in reinforcement learning, the learning is based on multiple episodes. These differences produce novel challenges in abandonment learning.

Also our work is much different from safe reinforcement learning, where we need to stay clear of catastrophic states, unlike the safe reinforcement learning, in our problem avoiding abandonment is not set as a hard constraint.

The paper is limited to less solutions to the learning threshold model. We are testing more strategies and we will be contrasting the best strategy.

We don't assume any prior incentive distribution in the UCB algorithm and so we have to rely on Hoeffding 's Inequality for a very simplified estimation. We 'd be able to make better bound predictions at Bayesian UCB in order to learn the distribution beforehand.

The Epsilon Greedy algorithm is the greediest of all algorithms for multi-armed bandit algorithms. We hold the value of the epsilon from 0 to 1. The higher the epsilon value the algorithm supports more exploration. In the beginning, we intend to adopt this approach to define the collection of behaviors depending on the consumer thresholds and then to minimize the importance of the epsilon to reduce the probability of failure and we strive to identify specific mechanisms in the custom policies.

In intermediate cases, the Thompson sampling is a more rational approach and lets one produce and classify more desirable tests. We sample one potential success score for each new contact from the beta distribution, which applies to each variant, and assign the contact to the variant with the lowest sampled success rate. The more data points we have collected, the more sure we can be regarding the actual success rate, and as we collect more evidence, the measured success rates will be steadily similar to the real result.

3.6 Why our Solution Is Better

Our solution is better because we are further exploring the learning strategies proposed in the paper by implementing more multi armed bandit strategies like epsilon greedy algorithm, Thompson sampling and Bayesian UCB algorithm.

In the UCB algorithm, we do not assume any prior on the reward distribution and therefore we have to rely on Hoeffding's Inequality for a very generalized estimation. In Bayesian UCB able to know the distribution upfront, we would be able to make better bound estimation

Epsilon Greedy algorithm is the greediest of all the multi armed bandit algorithms. We keep the value of epsilon between 0 and 1. The higher the value of epsilon the algorithm promotes more exploration. We aim to follow this strategy in the beginning to identify the set of actions based on the users thresholds and later on reduce the value of the epsilon to reduce the risk of abandonment and we aim to identify different structures in the personalized policies.

The Thompson sampling is a more principled approach and helps us to yield and identify more optimal results in marginal cases. We sample one possible success rate from the beta distribution that corresponds to each variant for each new contact, and assign the contact to the variant with the largest sampled success rate. The more data points we've observed, the more confident we'll be about the true success rate, and as we collect more data, the sampled success rates will be increasingly close to the real rate.

4. Hypothesis (or goals)

The goal is to develop a general threshold learning model for scenarios like personalizing the number of newsletters it sends to customers so that the user does not abandon the platform and discuss the structure of optimal policies.

We are implementing different strategies in the model and compare the performances among them

4.1 Single/multiple hypothesis

We are implementing different strategies in the model to develop a general threshold learning model for scenarios like personalizing the number of newsletters it sends to customers so that the user does not abandon the platform and discuss the structure of optimal policies.

Three different strategies have been implemented in the paper .They are

- UCB
- MOSS
- Explore exploit strategy

We are implementing three more :-

- Bayesian UCB
- Thompson Sampling

- ε-Greedy Algorithm

Our goal is to compare the performance of different strategies.

and compare the performances among them.

4.2 Positive or negative hypothesis

As Bayesian UCB, Epsilon greedy algorithm and Thompson sampling both fall under multi-armed bandit problem and each of these has different rates of discovery vs exploitation strategy both of these algorithms would operate in the case of abandonment learning, but the findings could vary greatly depending on the parameter values and could lead to several different outcomes. However they are guaranteed to generate results for each learning method from which we can draw suitable conclusions and equate them with the implementation of the author.

5. Methodology

5.1 How to generate/collect input data

The problem makes various assumptions for the input parameters like number of the users, the users thresholds and also about the distribution used for choosing an action at time t and and the reward function which produces the rewards for the platform based

on the actions performed by the user. All the assumptions made will be documented and will be added along with the code setup.

5.2 How to solve the problem

To solve the problem of a platform learning a set of actions and not risk the user abandoning the platform we try out three multi armed bandit algorithms which are

1. Bayesian UCB
2. Thompson Sampling
3. ε-Greedy Algorithm

By applying these algorithms we are trying to identify learning strategies for the platform without the user abandoning the platform in between. All these algorithms are explained in the coming section 5.2.1

5.2.1 Algorithm Design

The algorithms used in the project are:

1. Bayesian UCB

   This algorithm does not assume any prior reward functions and we depend on Hoeffding's Inequality for the reward function. The equation for Hoeffding's Inequality is given below.

Let $X_1,\ldots,X_t$ be i.i.d. (independent and identically distributed) random variables and they are all bounded by the interval [0, 1]. The sample mean is $\overline{Xt}$ = 1/t $\sum_{\tau=1}^{t} Xt$ Then for u > 0, we have:

$$\mathbb{P}[\mathbb{E}[X] > \overline{X}_t + u] \leq e^{-2tu^2}$$

Given one target action a, let us consider:

rt(a) as the random variables,

Q(a) as the true mean,

Q$\hat{}$ t(a) as the sample mean,

And u as the upper confidence bound, u=Ut(a)

Then we have,

$$\mathbb{P}[Q(a) > \hat{Q}_t(a) + U_t(a)] \leq e^{-2tU_t(a)^2}$$

We want to pick a bound so that with high chances the true mean is below the sample mean plus the upper confidence bound. Thus e−2tUt(a)2 should be a small probability. Let's say we are ok with a tiny threshold p:

$$e^{-2tU_t(a)^2} = p$$

Thus,

$$U_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}$$

2. Epsilon Greedy Algorithm

Most of the time, the ε-greedy algorithm takes the best action, but occasionally does random exploration. According to past experience, the action value is estimated by averaging the rewards associated with the target action a which we have observed to date (up to the current time step t):

$$\hat{Q}_t(a) = \frac{1}{N_t(a)} \sum_{\tau=1}^{t} r_\tau \mathbb{1}[a_\tau = a]$$

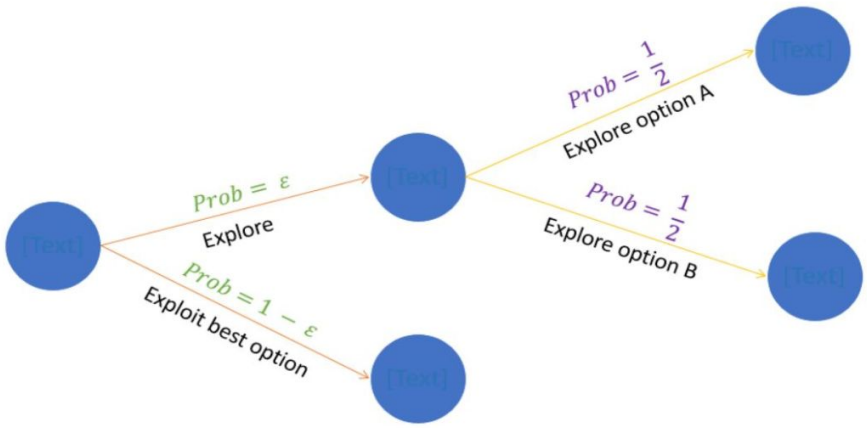
where ,

$\mathbb{1}$ is a binary indicator function

Nt(a) shows how many times the action a has been selected till then

$$N_t(a) = \sum_{\tau=1}^{t} \mathbb{1}[a_\tau = a].$$

For example, if we have a problem with two actions – A and B, the epsilon greedy algorithm works as shown below:



With probability 1- epsilon,we choose action with maximum value (argmaxa Qt(a))

With probability epsilon – we randomly choose an action from a set of all actions

3. Thompson Sampling

In Thompson sampling at each time step t, we select an action a according to the probability that a is optimal:

$$\pi(a \mid h_t) = \mathbb{P}[Q(a) > Q(a'), \forall a' \neq a \mid h_t]$$
$$= \mathbb{E}_{\mathcal{R}|h_t}[\mathbb{1}(a = \arg\max_{a \in \mathcal{A}} Q(a))]$$

where π(a|ht) is the probability of taking action a given the history ht.

It is normal to conclude that Q(a) follows a Beta distribution for the Bernoulli bandit, since Q(a) is basically the probability of success in Bernoulli distribution. The Beta(α, β) value is inside the interval [0 , 1] α and β refer to the counts when we have successfully or failed to get a reward respectively.

5.2.2 Language used

The Language used is python 3 and above. The libraries used for the project are

1. Matplotlib
2. Scipy
3. Numpy

4. Collections
5. Math
6. Random
7. Time

5.2.3 Tools used

The tools used in this project are

1. Jupyter Notebook
2. Anaconda Navigator

5.3 How to generate output

The output can be generated inline by running all the blocks of code in the jupyter notebook one after the other after installing all the required libraries.

The required libraries are:

1. Matplotlib
2. Scipy
3. Numpy
4. Collections
5. Math
6. Random
7. Time

5.4 How to test against hypotheses

We will validate the things learned from our algorithm against the authors work and observations. We will draw up meaningful conclusions based on our observations and see whether we were able to improve the learning strategies used by the platform. Since our model has no quantitative output we are unable to come up with exact quantitative measurements.

5.5 Proof of correctness

As Bayesian UCB, Epsilon greedy algorithm and Thompson sampling all comes under multi armed bandit problem and each of these has different level of exploration vs exploitation strategy all these algorithms will work in the case of learning with abandonment, however the results could vary significantly based on the parameter values and might lead to some different results. However they are guaranteed to produce patterns for each learning strategy from which we can make appropriate conclusions and compare it with the author's implementation.

6. Implementation

6.1 Code

Code in Appendices 10.1

6.2 Design Document And Flowchart

6.2.1 Design Document

6.2.1.1 Overview of the Design

The learning with abandonment system is developed in python3 using the jupyter notebook. The project implementation tries to understand the consequences that occur when a user abandons a platform in between and how the platform can learn from this and develop a series of actions that can effectively interact with the user without causing any displeasure or understanding the patience level of each user. The project tries to implement different multi armed bandit algorithms which comes under the reinforcement learning strategy and tries to learn different structures observed and make meaningful conclusions.

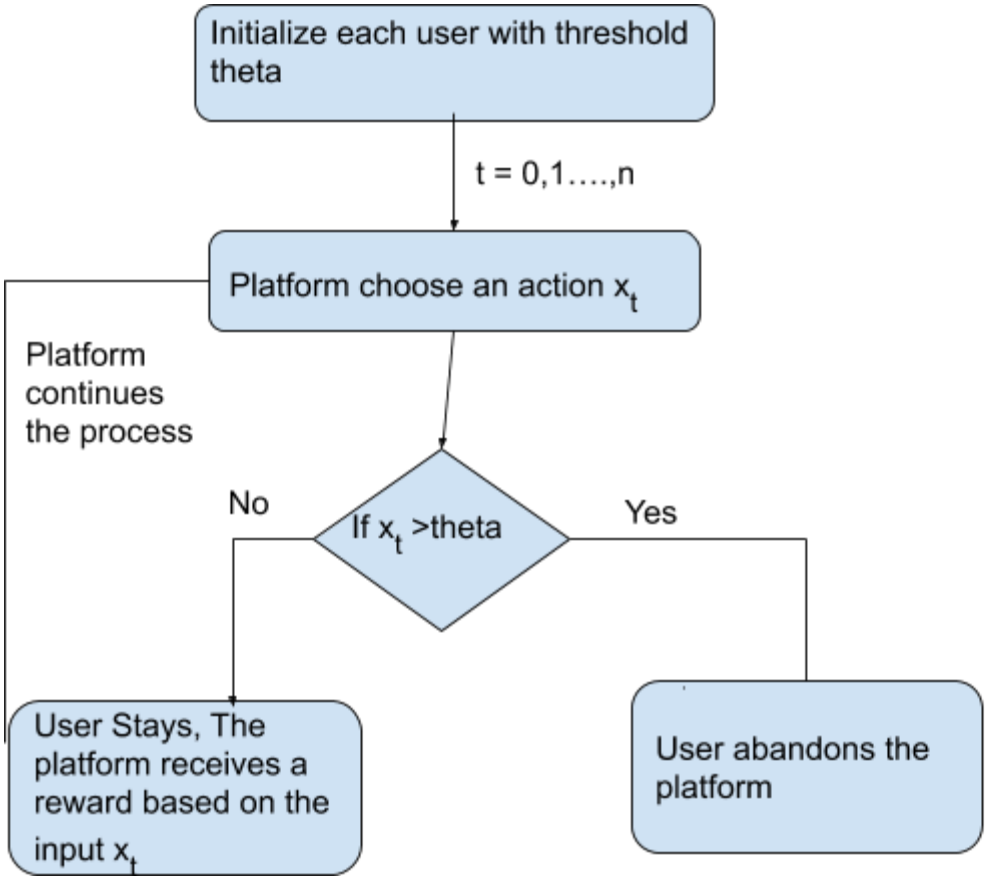
6.2.1.2 The Scope of the project

The scope of the job lies in understanding how each multi armed algorithm, namely Bayesian UCB, Epsilon greedy algorithm and Thompson sampling works under the predefined learning scenario of Learning Threshold works and draw meaningful conclusions. The conclusions are drawn from mean regrets produced by each different algorithm and analyse which algorithm gives the best performance.

6.2.1.3 Deliverables

The deliverables of our project is a jupyter notebook which shows all the outputs(graphs) we have generated as a part of the project implementation

6.2.2 Flowchart



7. Data Analysis And Discussion

7.1 Output Generation

We have implemented 3 multi armed bandit strategies and used a simulation setting for obtaining the output. In the simulation we are computing the best action value and the value obtained when each of the three multi armed bandit strategies is used. We compare the obtained results with the best action value to calculate the mean regret observed for each of the strategies.

We are giving the three policies as input to our abandonment simulation method and calculate the mean regret for each policy by comparing it with the best action value obtained in each of the cases.

Finally to have a better visualization of the outputs we plot the mean regrets as graphs where the x-axis represents the time t, at which a platform performs an action and y-axis represents the regret obtained in each of the cases. We also output the mean regret observed in each of the strategies.

7.2 Output Analysis

All the three graphs are analysed .Cumulative regret paths with regret on y axis and time on x axis are plotted .

Out of three strategies implemented it can be seen that epsilon greedy algorithm has least regret of 42.9

In the Epsilon-Greedy algorithm random selection between exploration and exploitation happens to balance exploration and exploitation.

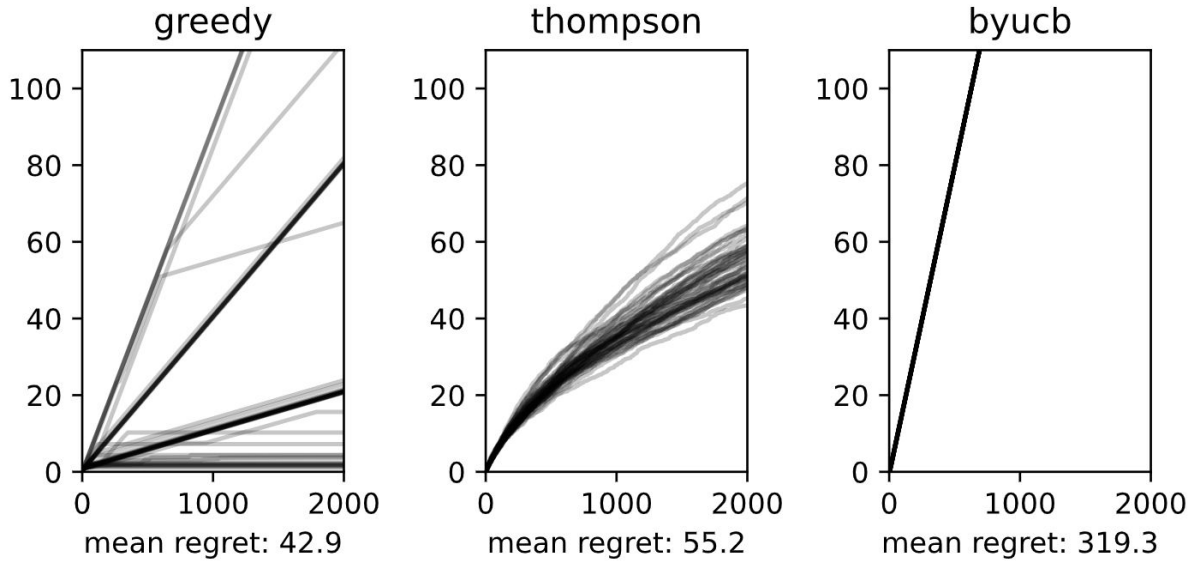
In Bayesian UCB prior information about the distribution of rewards unlike UCB algorithm.We are getting regret=319.3 for Bayesian UCB .

In Thompson sampling at each time step t, we select an action a according to the pr probability that a is optimal.

We are getting regret =55.2 for Thompson Sampling.

Out of the three strategies we have implemented ,Epsilon greedy has least regret and Bayesian UCB has the highest regret.

Please find below the graphs for different strategies.



7.3 Compare Output Against Hypothesis

Our hypothesis was to implement different strategies in the model and compare the performances among them.From the output analysis we see that epsilon greedy has least regret .Bayesian UCB has the highest regret of 319.3 and Thompson UCB has a regret of 55.2.

7.4 Discussion

The three strategies which have been mentioned in the research paper are UCB ,MOSS and explore exploit strategy.UCB has mean regret of 104.9,MOSS has mean regret of 79.7 and explore exploit has mean regret of 36.4.The strategies which we have

implemented are Epsilon greedy with mean regret of 42.9 ,Thompson Sampling with mean regret of 55.2 and Bayesian UCB with mean regret of 319.3.Out of all the six strategies explore exploit has the least mean regret.

8. Conclusions And Recommendations

8.1 Summary And Conclusions

Our project revolves around different multi armed bandit algorithms which come under the reinforcement learning algorithms to understand when a user will abandon a platform and how the platform should continue its learning process/optimizations for serving the future users better and prevent users from abandoning the platform.

We have tried three multi armed bandit algorithms to understand which algorithm gives better performance. The performance of the algorithm is based primarily on the mean regret values produced by the algorithm and graphs are plotted against the mean regrets.

From our research we came to the conclusion that Epsilon greedy algorithm produced the least mean regret while bayesian UCB algorithm produced the highest mean regret. We choose the mean regret to measure the performance of the algorithms because it shows how much an algorithm deviates from the actual value.

From the authors implementation and our own research we have come to the conclusion that generally the explore exploit strategy produces better results when compared to other strategies when performance is measured based on the least mean regret produced by the algorithm.

8.2 Recommendations For Future Studies

The recommendation for future studies which are outside the scope of our project are:

8.2.1 Abandonment models

Firstly, attention should be extended to more nuanced device abandonment behaviour. This might take several ways, including a cumulative budget for resilience that gets drained when the threshold is reached. Another paradigm is that of a user themselves performing a learning game, contrasting this method to one or more external alternatives. In this case, both the customer and the application think about one another at the same time.

8.2.2 User information

Second , in terms of covariates, we did not allow additional user knowledge. User behavior seems like a major indication of her desires in the notification case. Models

capable of integrating such knowledge and capable of inferring the data parameters are outside the scope of this study but are a significant path for future analysis.

8.2.3 Empirical analysis

This work focuses on the theoretical interpretation of the concept of abandonment, while missing essential facets of a structure in real life. We assume there is considerable scope to obtain more information from an observational viewpoint using real-world structures with possibility of abandonment.

9. Bibliography

[1]Johari, Ramesh, and Sven Schmit. “Learning with Abandonment.” Accessed August 28, 2020. https://arxiv.org/pdf/1802.08718v1.pdf.

[2]Schmit, Sven. “GitHub - Schmit/Learning-Abandonment: Repo Accompanying Learning with Abandonment.” *GitHub*, https://github.com/schmit/learning-abandonment. Accessed 28 Aug. 2020.

[3]Galbraith, Byron. “GitHub - Bgalbraith/Bandits: Python Library for Multi-Armed Bandits.” GitHub. Accessed August 28, 2020. https://github.com/bgalbraith/bandits.

[4]Bubeck, Sébastien. “Bandit Theory, Part I | I’m a Bandit.” I’m a bandit, May 11, 2016. https://blogs.princeton.edu/imabandit/2016/05/11/bandit-theory-part-i/.

[5]I’m a bandit. “Bandit Theory, Part II | I’m a Bandit,” May 13, 2016. https://blogs.princeton.edu/imabandit/2016/05/13/bandit-theory-part-ii/.

10. Appendices

10.1 Program Source Code With Documentation

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import collections, math, random, time

import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
get_ipython().run_line_magic('config', "InlineBackend.figure_format = 'svg'")


from scipy.stats import norm

import numpy as np


# In[2]:


def bernoulli(p): return 1*(random.random() < p)

def plogpq(p, q):
    if p == 0:
        return 0
    if q == 0:
        return math.inf
    return p * math.log(p/q)
```

```
def kldiv(p, q):
    return plogpq(p, q) + plogpq(1-p, 1-q)

def newton(f, fp, x, eps=1e-5, projection=lambda x: x):
    while True:
        xn = projection(x - f(x)/fp(x))
        print(xn)
        if abs(xn - x) < eps:
            return xn
        x = xn

def proj_unit(x): return max(0, min(1, x))
def bisect_fn(fn, lowerbound, upperbound, eps=1e-4):
    if fn(lowerbound) > 0:
        return bisect_fn(lambda x: -fn(x), lowerbound, upperbound)

    assert fn(upperbound) > 0, "LB and UB must have opposing signs"

    while abs(upperbound - lowerbound) > eps:
        midpoint = lowerbound/2 + upperbound/2
        if fn(midpoint) > 0:
            upperbound = midpoint
        else:
            lowerbound = midpoint

    return lowerbound/2 + upperbound/2


# In[3]:


History = collections.namedtuple("History", ['s','n'])
#Our history array
Hstry = []
count = [0]
```

```
def update_history(history, outcome):
    Hstry.append(outcome)
    count[0] += 1
    return History(history.s + outcome, history.n + 1)

def mean(history):
    if history.n > 0:
        return history.s / history.n
    return 0

def ucb_index(history, t, K, alpha=2):
    if history.n == 0: return 1e10 + random.random()
    return mean(history) + math.sqrt(alpha * math.log(t) / (2 * history.n))

def byucb_index(history, t, K, c=1.5,hstry=Hstry):
    if history.n == 0: return 1e10 + random.random()
    std = np.std(history.n)
    return (c*std)/math.sqrt(history.n/t)

def moss_index(history, t, K):
    if history.n == 0: return 1e10 + random.random()
    return mean(history) + math.sqrt(max(math.log(t/(K*history.n)), 0) /
history.n)

def greedy_index(history, t, K):
    if history.n == 0: return 1e10 + random.random()
    return mean(history)

def thompson_index(history, t, K, a=1, b=1):
    return random.betavariate(a + history.s, b + history.n - history.s)


def klucb_index(history, t, K, c=0):
    if history.n == 0: return 1e10 + random.random()
    phat = mean(history)

    if phat == 1:
```

```
        return 1

    f = lambda q: kldiv(phat, q) - (math.log(t) + c * math.log(math.log(t))) /
history.n

    # Based on bisection
    return bisect_fn(f, phat, 1)

    # Based on Newton's method
    # fp = lambda q: -phat/max(1e-5, q) + (phat-1)/max(q-1, 1e-5)
    # return newton(f, fp, phat/2 + 1/2, projection=proj_unit)


def index_policy(histories, index, *args):
    t = sum(history.n for history in histories) + 1
    K = len(histories)
    return max(range(K), key=lambda i: index(histories[i], t, K, *args))

def ucb(histories): return index_policy(histories, ucb_index)
def moss(histories): return index_policy(histories, moss_index)
def greedy(histories): return index_policy(histories, greedy_index)
def thompson(histories): return index_policy(histories, thompson_index)
def klucb(histories): return index_policy(histories, klucb_index)
def byucb(histories): return index_policy(histories, byucb_index)


# In[4]:


def max_concave(f, xmin=0, xmax=1, eps=1e-10):
    """ find the maximum of a 1d concave function on a bdd interval"""
    m = xmin/2 + xmax/2
    if m - xmin < eps:
        return m

    fmin, fm, fmax = f(xmin), f(m), f(xmax)
```

```
    if fm > fmin and fm > fmax:
        return max((max_concave(f, m, xmax), max_concave(f, xmin, m)),
                   key=lambda x: f(x))

    if fm < fmin:
        return max_concave(f, xmin, m)

    return max_concave(f, m, xmax)


# In[5]:


def bandit(policy, pull, K, T):
    histories = [History(0, 0) for _ in range(K)]
    actions = []

    for t in range(T):
        action = policy(histories)
        outcome = pull(action)
        histories[action] = update_history(histories[action], outcome)
        actions.append(action)

    return actions, histories

def best_action_value(F, r):
    p = lambda x: r(x) * (1-F(x))

    xstar = max_concave(p)
    M = p(xstar)
    return xstar, M

def abandonment_pulls(K, F, r):
    def pull(action):
        x = (action + 1)/K
        return r(x) * bernoulli(1-F(x))
```

```
    return pull
def listify(gen):
    "Convert a generator into a function which returns a list"
    def patched(*args, **kwargs):
        return list(gen(*args, **kwargs))
    return patched

@listify
def abandonment_regret(actions, F, r, M=0):
    p = lambda x: r(x) * (1-F(x))

    R = 0
    for action in actions:
        R += M - p(action)
        yield R


# In[6]:


def plot_regrets(regrets, ax=None, title="Regret plot", style="k-"):
    if ax is None:
        f, ax = plt.subplots()

    nrep = len(regrets)
    T = len(regrets[0])

    for R in regrets:
        line, = ax.plot(R, style, alpha=0.2 + 1/nrep)

    line.set_label(title)

    ax.set_title(title)
    ax.set_xlabel("mean regret: {:.1f}".format(sum(R[-1] for R in regrets)/nrep))
    ax.set_xlim(0, T)
```

```
def plot_actions(actions, ax=None):
    if ax is None:
        f, ax = plt.subplots()

    ax.imshow(actions.T, aspect="auto")
    ax.tick_params(axis='both',
                   which='both',
                   bottom='off',
                   top='off',
                   left='off',
                   labelbottom='off',
                   labelleft='off')

# In[7]:

def abandonment_simulation(T, K, F, r, policy, nrep=1, action_agg=20):
    pull = abandonment_pulls(K, F, r)
    regrets = []
    action_matrix = np.zeros((T//action_agg, K))

    xstar, M = best_action_value(F, r)

    for _ in range(nrep):
        actions, histories = bandit(policy, pull, K, T)
        R = abandonment_regret([(i+1)/K for i in actions], F, r, M)

        # record regret
        regrets.append(R)
        # record actions
        for t, a in enumerate(actions):
            action_matrix[t//action_agg][a] += 1
```

```
    return regrets, action_matrix

# In[8]:

import heapq
import bisect

import numpy as np

# In[9]:

class ECDF:
    def __init__(self):
        self.data = []

    def observe(self, x):
        bisect.insort(self.data, x)

    def evaluate(self, x):
        n = len(self.data)
        if n == 0:
            return 0.5
        return bisect.bisect(self.data, x) / n

    def ecdf(self):
        n = len(self.data)
        if n < 200:
            return ((x, i/n) for i, x in enumerate(self.data))
        return ((x, self.evaluate(x))
                for x in np.linspace(self.data[0], self.data[-1], 100))
    def plot(self):
        x, e = zip(*self.ecdf())
```

```
    f, ax = plt.subplots()
    ax.plot(x, e)
    ax.set_title("Empirical CDF")
    ax.set_xlabel(r"$x$")
    ax.set_ylabel(r"$\hat F(x)$")


# In[10]:


def oracle(T, r, sampler=random.random):
  ecdf = ECDF()
  actions = []
  for _ in range(T):
    action = max(np.linspace(0, 1, random.randint(100, 200)),
                 key=lambda x: r(x) * (1-ecdf.evaluate(x)))
    threshold = sampler()

    # cheat
    ecdf.observe(threshold)
    actions.append(action)

  return actions, ecdf

def oracle_simulations(oracle, T, F, r, sampler, nrep=1):
  _, M = best_action_value(F, r)

  regrets = []
  for _ in range(nrep):
    actions, ecdf = oracle(T, r, sampler=sampler)
    R = abandonment_regret(actions, F, r, M)
    regrets.append(R)

  return regrets
def explore_exploit(T, r, alpha=1/2, sampler=random.random):
  ecdf = ECDF()
```

```
    actions = []
    for t in range(T):
        if t <= 20+2*T**alpha:
            action = 0
            threshold = sampler()
            ecdf.observe(threshold)
        else:
            action = action = max(np.linspace(0, 1, random.randint(100, 200)),
                                  key=lambda x: r(x) * (1-ecdf.evaluate(x)))
        actions.append(action)

    return actions, ecdf


# In[11]:


r = lambda x: x
F = lambda x: x
sampler = lambda: random.random()

xstar, M = best_action_value(F, r)

f, ax = plt.subplots(1, 3, figsize=(8, 3))

T = 2000
K = int(2 * max(1, (T / math.log(T))**0.25 + 1))
print("K = {}".format(K))

nrep = 50

# algos
for i, policy in enumerate([ucb, klucb]):
    print(policy)
    regrets, actions = abandonment_simulation(T, K, F, r, policy, nrep=nrep)
    plot_regrets(regrets, ax=ax[i], title=policy.__name__)
```

```
# explore-exploit
regrets = oracle_simulations(explore_exploit, T, F, r, sampler, nrep)
plot_regrets(regrets, ax=ax[-1], title="explore-exploit")

for a in ax:
    a.set_ylim(0, 110)
    a.set_xlim(0, T)

f.tight_layout()


# In[12]:


r = lambda x: x
F = lambda x: x
sampler = lambda: random.random()

xstar, M = best_action_value(F, r)

f, ax = plt.subplots(1, 4, figsize=(8, 3))

T = 2000
K = int(2 * max(1, (T / math.log(T))**0.25 + 1))
print("K = {}".format(K))

nrep = 50

# algos
#moss,greedy,
for i, policy in enumerate([moss,greedy,thompson,byucb]):
    print(policy)
    regrets, actions = abandonment_simulation(T, K, F, r, policy, nrep=nrep)
    plot_regrets(regrets, ax=ax[i], title=policy.__name__)

for a in ax:
    a.set_ylim(0, 110)
```

```
    a.set_xlim(0, T)

f.tight_layout()


# In[13]:


def linear_grid(a, b, steps):
    # this is an interior linear grid that does not contain a or b
    stepsize = (b-a)/steps
    return [round(a + (i+0.5) * stepsize, 3) for i in range(steps)]

# value iteration operator
def l2_dist(a, b):
    return sum((v - b[x][0])**2 for x, (v, _) in a.items()) / len(a)

def vi(initial, operator, converged):
    def _vi(old, new):
        if converged(old, new):
            return new
        return _vi(new, operator(new))

    return _vi(initial, operator(initial))


# In[14]:


# some distribution functions
def Funif(x):
    return max(min(x, 1), 0)

def Fbetab(x, b):
    return 1 - (1-x)**b

# some reward functions
linear = lambda x: x
```

```
neglog = lambda x: -math.log(1-x)

steps = 201


# In[15]:


def Vop(F, grid, r=lambda x: x, beta=1, discount=0.9):
  def Fcond(y, a, b):
    nom = F(y) - F(a)
    denom = F(b) - F(a)
    if denom > 0:
      return nom / denom
    return 0

  def _Vop(xvx):
    def v(y, a, b):
      prob_success = 1-Fcond(y, a, b)
      val_success = r(y) + discount * xvx[(y, b)][0]
      val_fail = beta * discount * xvx[(a, y)][0]
      return prob_success * val_success + (1-prob_success) * val_fail

    def Vnew(state):
      a, b = state
      return max(((v(y, a, b), y) for y in grid if a <= y < b), default=(0, 0))

    return {state: Vnew(state) for state, Vstate in xvx.items()}

  return _Vop


# In[16]:


def analyze_vopt(F=Funif, grid=linear_grid(0, 1, steps), reward_fn=linear,
beta=1, discount=0.8, tol=1e-3):
```

```
    vopt = compute_vopt(F, grid, reward_fn, beta, discount, tol)
#     _ = plot_vopt(vopt, grid, discount)
    return vopt

def compute_vopt(F=Funif, grid=linear_grid(0, 1, steps), reward_fn=linear,
beta=1, discount=0.8, tol=1e-5):
    initial = {(x, y): (0, 0) for x in grid for y in grid if x <= y}
    return vi(initial, Vop(F, grid, reward_fn, beta=beta, discount=discount),
lambda x, y: l2_dist(x, y) < tol)


# In[17]:


# extracting optimal actions

grid = linear_grid(0, 1, steps)
lb, ub = grid[0], grid[-1]

def action(vopt, lower, upper):
    v, y = vopt[(lower, upper)]
    return y

def action_tree(vopt, init_lower, init_upper):
    y = action(vopt, init_lower, init_upper)
    if y == init_lower:
        return y
    return y, action_tree(vopt, init_lower, y), action_tree(vopt, y, init_upper)

# code to plot the action tree

def left(tree):
    return tree[1]

def right(tree):
    return tree[2]

def value(tree):
```

```
    if isinstance(tree, tuple):
        return tree[0]
    return tree


# In[18]:


def plot_line(axes, a, b, style="r-", alpha=1.0, x_offset=0):
    axes.plot([a[0]+x_offset, b[0]-x_offset], [a[1], b[1]], style, alpha=alpha)
    axes.plot([a[0]], [a[1]], "k. ")


def plot_node(axes, tree, index=0, depth=10, x_offset=0):
    if isinstance(tree, tuple) and index < depth:
        plot_line(axes, (index, value(tree)), (index+1, value(left(tree))), style="r--",
x_offset=x_offset)
        plot_line(axes, (index, value(tree)), (index+1, value(right(tree))),
style="g-", x_offset=x_offset)
#         axes.plot([index, index+1], [value(tree), value(left(tree))],
color="darkred")
#         axes.plot([index, index+1], [value(tree), value(right(tree))],
color="steelblue")
        plot_node(axes, left(tree), index+1, depth)
        plot_node(axes, right(tree), index+1, depth)
    else:
        plot_line(axes, (index, value(tree)), (depth, value(tree)), style="k:",
alpha=0.8, x_offset=x_offset)
#         axes.plot([index, depth], [value(tree), value(tree)],
#                   color="black", opacity=0.5)

def plot_tree(tree, width=6, height=4, depth=10):
    f, ax = plt.subplots(figsize=(width, height))

    ax.set_ylim(0, 1)
    ax.set_title("Visualization of the optimal policy tree")
```

```
    ax.set_xlabel("time step t")
    ax.set_ylabel("action at time t x(t)")

    for i in range(depth+1): ax.axvline(i, color="black", alpha=0.1)

    plot_node(ax, tree, depth=depth)
    return f, ax

def plot_actions(vopt, lb, ub, depth=10):
    tree = action_tree(vopt, lb, ub)
    c, a = plot_tree(tree, depth=depth)
    return c, a

def plot_vopt(vopt, grid, discount):
    vopt_list = [(a, b, v) for (a, b), (v, y) in vopt.items()]

    canvas = tp.Canvas(500, 400)
    axes = canvas.cartesian()


    axes.plot([0, 1], [0, 1])

    for lower in grid:
        sublist = [(b, (1-discount)*v) for a, b, v in vopt_list if a == lower and b !=
lower]
        try:
            X, V = zip(*sublist)

            axes.plot(X, V, color="black", opacity=0.5)
        except ValueError:
            pass

    return canvas, axes


# In[19]:
```

```
vopt = analyze_vopt(discount=0.9, beta=0.5)
f, ax = plot_actions(vopt, lb, ub)

# In[20]:

discount = 0.8
beta_path = [(beta,
              compute_vopt(Funif,
                           linear_grid(0, 1, steps),
                           reward_fn=linear,
                           beta=beta,
                           discount=discount,
                           tol=1e-2))
  for beta in linear_grid(0, 1, 31)]

action_path = [(beta, action(vopt, lb, ub)) for beta, vopt in beta_path]
value_path = [(beta, vopt[(lb, ub)][0]) for beta, vopt in beta_path]

# In[21]:

f, ax = plt.subplots(figsize=(6, 4))
ax.set_title("First action in relation to override probability")
ax.set_xlabel("probability of override")
ax.set_ylabel(r"$x_0$: optimal action at time 0")

ax.set_xlim(0, 1)

x, y = zip(*action_path)
ax.plot([0] + list(x) , [0.5] + list(y), "k. ")
ax.axhline(0.5, color="k", alpha=0.1)

# In[ ]:
```

10.2 Input/Output Listing

Input-

We give the following input parameters and plot the cumulative regret paths

K-number of arms

T-Time steps

N-number of repetitions

We are giving the below values for them:-

n=50

T=2000

K=12

Output -

Cumulative regret paths with regret on y axis and time on x axis are plotted .

As discussed in the document above(Section 7) we get the following regrets

Mean regret of Epsilon greedy algorithm =42.9,thompson sampling =55.2 and Bayesian UCB =319.3

10.3 Other Related Material

The other related materials are

1. Dynamic Learning of Sequential Choice Bandit Problem under Marketing Fatigue

   Authors:

   Junyu Cao

   Wei Sun

2. Potential Good Abandonment Prediction

   Authors:

   Aleksandr Chuklin

   Pavel Serdyukov

3. Measuring the reliability of reinforcement learning algorithms

   Authors:

   Stephanie C.Y. Chan

   Samuel Fishman

   John Canny

   Anoop Korattikara

   Sergio Guadarrama