AN APPLICATION OF REINFORCEMENT LEARNING TO CREDIT SCORING BASED ON THE LOGISTIC BANDIT FRAMEWORK

Mr. Kantapong Visantavarakul

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Statistics
Department of Statistics
FACULTY OF COMMERCE AND ACCOUNTANCY
Chulalongkorn University
Academic Year 2022
Copyright of Chulalongkorn University

การประยุกต์ใช้การเรียนรู้แบบเสริมกำลังสำหรับการให้คะแนนเครดิตภายใต้กรอบปัญหาโลจิสติก
แบนดิต

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

นายกันตพงษ์ วิสารทวรากูล

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาสถิติ ภาควิชาสถิติ
คณะพาณิชยศาสตร์และการบัญชี จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2565
ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

Thesis Title	AN APPLICATION OF REINFORCEMENT LEARNING TO CREDIT SCORING BASED ON THE LOGISTIC BANDIT FRAMEWORK
By	Mr. Kantapong Visantavarakul
Field of Study	Statistics
Thesis Advisor	Associate Professor SEKSAN KIATSUPAIBUL, Ph.D.

Accepted by the FACULTY OF COMMERCE AND ACCOUNTANCY, Chulalongkorn University in Partial Fulfillment of the Requirement for the Master of Science

.... Dean of the FACULTY OF COMMERCE AND ACCOUNTANCY
(Associate Professor WILERT PURIWAT, Ph.D.)

THESIS COMMITTEE

.... Chairman
(Associate Professor VITARA PUNGPAPONG, Ph.D.)

.... Thesis Advisor
(Associate Professor SEKSAN KIATSUPAIBUL, Ph.D.)

.... Examiner
(Assistant Professor NUTTIRUDEE CHAROENRUK, Ph.D.)

.... External Examiner
(Associate Professor SUNTI TIRAPAT, Ph.D.)

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

กันตพงษ์ วิสารทวรากูล : การประยุกต์ใช้การเรียนรู้แบบเสริมกำลังสำหรับการให้คะแนนเครดิตภายใต้กรอบปัญหาโลจิสติกแบนดิต. (AN APPLICATION OF REINFORCEMENT LEARNING TO CREDIT SCORING BASED ON THE LOGISTIC BANDIT FRAMEWORK) อ.ที่ปรึกษาหลัก : รศ. ดร.เสกสรร เกียรติสุไพบูลย์

งานวิจัยนี้มีวัตถุประสงค์เพื่อประยุกต์ใช้การเรียนรู้แบบเสริมกำลังสำหรับการให้คะแนนเครดิตภายใต้กรอบปัญหาโลจิสติกแบนดิต การให้คะแนนเครดิตและการให้สินเชื่อสามารถจัดอยู่ในรูปแบบปัญหาการตัดสินใจอย่างเป็นลำดับโดยผู้ให้สินเชื่อจะตัดสินใจเลือกการกระทำโดยที่จุดสิ้นสุดของเวลานั้นไม่มีกำหนด วิธีการให้คะแนนเครดิตแบบดั้งเดิมพิจารณาการสร้างโมเดลแยกออกจากการให้สินเชื่อ ในการเรียนรู้แบบเสริมกำลัง วิธีนี้เรียกว่า ขั้นตอนวิธีแบบละโมบ (greedy algorithm) ซึ่งเชื่อกันอย่างแพร่หลายว่าให้ประสิทธิภาพที่ด้อยกว่าการเรียนรู้แบบเสริมกำลังที่มีประสิทธิภาพ เช่น การสุ่มตัวอย่างแบบทอมสัน (Thompson sampling) สมมติฐานนี้เป็นจริงในสถานการณ์แบบง่าย นั่นคือ ในแต่ละช่วงเวลาผู้ให้กู้จะให้สินเชื่อได้แค่คนเดียวในขณะที่ผู้กู้สินเชื่อยังเป็นรายเดิมอยู่ตลอด อย่างไรก็ตาม ในสถานการณ์ที่สมจริงมากขึ้น นั่นคือ ไม่มีเงื่อนไขทั้งสองข้อดังกล่าว ขั้นตอนวิธีแบบละโมบสามารถให้ประสิทธิภาพที่ดีกว่าการสุ่มตัวอย่างแบบทอมสัน เนื่องจาก ขั้นตอนวิธีแบบละโมบไม่ได้ยึดติดเร็วเกินไปกับการกระทำที่ให้ผลตอบแทนที่ด้อยกว่าซึ่งไม่เหมือนกับในสถานการณ์แบบง่าย ถึงแม้จะเป็นเช่นนั้น การสำรวจแบบมีประสิทธิภาพของการสุ่มตัวอย่างแบบทอมสันก็ยังมีประโยชน์ในการเรียนรู้ภายใต้สถานการณ์นี้ ในกรณีที่จำนวนตัวแปรที่อธิบายลักษณะของผู้ขอกู้สินเชื่อมีจำนวนมาก การสุ่มตัวอย่างแบบทอมสันสามารถให้ผลลัพธ์ที่ดีกว่าขั้นตอนวิธีแบบละโมบ ผลลัพธ์ที่ได้จากการศึกษานี้คาดว่าจะเป็นประโยชน์ในการทำความเข้าใจการเรียนรู้แบบเสริมกำลังภายใต้กรอบปัญหาโลจิสติกแบนดิตได้ดียิ่งขึ้น โดยเฉพาะในกระบวนการให้คะแนนเครดิตและการให้สินเชื่อ

สาขาวิชา สถิติ	ลายมือชื่อ นิสิต ....
ปีการศึกษา 2565	ลายมือชื่อ อ.ที่ปรึกษาหลัก ....

6480388626 : MAJOR STATISTICS
KEYWORD: REINFORCEMENT LEARNING, CREDIT SCORING, LOGISTIC BANDIT, GREEDY ALGORITHM, THOMPSON SAMPLING

Kantapong Visantavarakul : AN APPLICATION OF REINFORCEMENT LEARNING TO CREDIT SCORING BASED ON THE LOGISTIC BANDIT FRAMEWORK. Advisor: Assoc. Prof. SEKSAN KIATSUPAIBUL, Ph.D.

This study applies reinforcement learning to credit scoring by using the logistic bandit framework. The credit scoring and the credit underwriting are modeled into a single sequential decision problem where the credit underwriter takes a sequence of actions over an indefinite number of time steps. The traditional credit scoring approach considers the model construction separately from the underwriting process. This approach is identified as a greedy algorithm in the reinforcement learning literature, which is commonly believed to be inferior to an efficient reinforcement learning approach such as Thompson sampling. This is true under the simple setting, i.e., granting credit to a single borrower per action while the pool of the borrowers is fixed. However, under the more realistic scenario where these two conditions are relaxed, the greedy approach can outperform Thompson sampling since the greedy algorithm does not commit too early to an inferior action as it does in the simple setting. Still, the efficient exploration feature of Thompson sampling is beneficial. When the borrower characteristics are captured by a large number of features, the exploration mechanism enables Thompson sampling to outperform the greedy algorithm. The results from the simulation study permit a deeper understanding of the reinforcement learning approaches towards the logistic bandits, especially in the setting of credit scoring and credit underwriting processes.

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

Field of Study:	Statistics	Student's Signature ....
Academic Year:	2022	Advisor's Signature ....

ACKNOWLEDGEMENTS

Regarding the thesis "AN APPLICATION OF REINFORCEMENT LEARNING TO CREDIT SCORING BASED ON THE LOGISTIC BANDIT FRAMEWORK", I would like to offer special thanks to my advisor, Assoc. Prof. Seksan Kiatsupaibul, Ph.D., for inspiring my interest in reinforcement learning and for the close guidance in each step of the research, from setting up the research topic to giving advices on revising and proofreading the paper. Due to his expertise in Mathematics and Finance, I was able to successfully complete this research.

I would like to thank committee members, namely Assoc. Prof. Vitara Pungpapong, Ph.D., Asst. Prof. Nuttirudee Charoenruk, Ph.D. and Assoc. Prof. Sunti Tirapat, Ph.D., for giving insightful comments and practical suggestions on the study. Their comments and suggestions led to considerable improvements in this research.

Finally, I would like to mention my family for their patience and emotional support during this process. The motivations from my friends and seniors helped me in this process as well.

Kantapong Visantavarakul

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

TABLE OF CONTENTS

Page

ABSTRACT (THAI) iii
ABSTRACT (ENGLISH) iv
ACKNOWLEDGEMENTS v
TABLE OF CONTENTS vi
LIST OF TABLES viii
LIST OF FIGURES ix
CHAPTER I INTRODUCTION 1
1.1 Background and Rationale 1
1.2 Objectives 3
1.3 Scope of Study 4
1.4 Expected Benefits 4
CHAPTER II RELATED WORKS 5
2.1 Logistic Regression 5
2.2 Bernoulli Bandit 5
2.3 Logistic Bandit 9
2.4 Greedy Algorithm 10
2.5 Thompson Sampling 11
2.6 Laplace Approximation 11
2.7 Metropolis Hasting 12
2.8 Langevin Monte Carlo Markov Chain (Langevin MCMC) 13
CHAPTER III METHODOLOGY 15
3.1 Logistic Bandit Framework and Simulation Method 15
3.2 Performance Measures 18
3.3 Reinforcement Learning Algorithms 19
3.4 Algorithm Flowchart 24

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

CHAPTER IV RESULTS 25
4.1 Small Number of Features ($p = 2$) 26
4.2 Medium Number of Features ($p = 10$) 30
4.3 Large Number of Features ($p = 20$) 35
CHAPTER V CONCLUSION AND DISCUSSION 41
5.1 Conclusion 41
5.2 Discussion 44
5.3 Future Research 48
REFERENCES 49
Appendix 1: Python code for Bernoulli bandit 50
A1.1 Package dependencies 50
A1.2 Bernoulli bandit environment 50
A1.3 Greedy and epsilon greedy algorithms 50
A1.4 Thompson sampling algorithm and Upper Confidence Bound algorithm 51
A1.5 Simulation 52
A1.6 Visualizations 53
Appendix 2: Python code for logistic bandit 56
A2.1 Dependencies 56
A2.2 Logistic bandit environment 56
A2.3 Greedy and epsilon-greedy algorithms 58
A2.4 Thompson sampling with Laplace / Langevin MCMC 60
A2.5 Simulation 63
A2.6 Visualizations 65
VITA 68

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

LIST OF TABLES

Page

Table 1. The comparison between traditional logistic bandit and credit scoring 15

Table 2. The cumulative rewards of each algorithm under different feature dimensions 42

Table 3. The cumulative regrets of each algorithm under different feature dimensions 43

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

LIST OF FIGURES

Page

Figure 1. Performance comparisons on Bernoulli bandit 7

Figure 2. Performances of epsilon-greedy ($\varepsilon = 0.01, 0.05, 0.1$) on Bernoulli bandit 8

Figure 3. The interaction between agent and environment in logistic bandit framework 17

Figure 4. Histogram of sample average signal-to-noise ratio and sample average non-default probability under a small (2) and a large (20) number of features 18

Figure 5. Algorithm flowchart 24

Figure 6. Performance comparisons on two-dimensional features (single borrower without renewal) 26

Figure 7. Performance comparisons on two-dimensional features (single borrower with renewal) 27

Figure 8. Performance comparisons on two-dimensional features (multiple borrowers without renewal) 29

Figure 9. Performance comparisons on two-dimensional features (multiple borrowers with renewal) 30

Figure 10. Performance comparisons on ten-dimensional features (single borrower without renewal) 31

Figure 11. Performance comparisons on ten-dimensional features (single borrower with renewal) 32

Figure 12. Performance comparisons on ten-dimensional features (multiple borrowers without renewal) 33

Figure 13. Performance comparisons on ten-dimensional features (multiple borrowers with renewal) 34

Figure 14. Performance comparisons on twenty-dimensional features (single borrower without renewal) 36

Figure 15. Performance comparisons on twenty-dimensional features (single borrower with renewal) 37

Figure 16. Performance comparisons on twenty-dimensional features (multiple borrowers without renewal) 38

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

Figure 17. Performance comparisons on twenty-dimensional features (multiple borrowers with renewal) 39

Figure 18. Non-default probability, signal-to-noise ratio and percentage improvement across feature dimensions 46

Figure 19. Performance of Thompson sampling with Langevin MCMC when initial parameter is deviated on twenty-dimensional features (multiple borrowers with renewal).. 47

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

CHAPTER I
INTRODUCTION

1.1 Background and Rationale

Credit risk is an uncertainty regarding whether the credit condition of a borrower worsens to a degree that the borrower could not repay the debt in a full amount. If a borrower defaults on a loan, the lender has to write off all or most of remaining balances, especially when the loan is unsecured. Compared with other types of risk, i.e. market risk, liquidity risk and operational risk, credit risk is the most important factor in deciding how loan should be valued. (Phillips, 2018) In the context of credit risk, three quantities need to be considered, i.e. probability of non-default, loss given default and exposure at default. However, the most important quantity is the probability of non-default, defined as the probability that a borrower would pay back the loan amount in full. In the context of individual borrowers, one main reason of defaulting on loan is bad individual decision in applying for a loan or financial mismanagement after a loan is taken. To estimate the probability of non-default, modern approaches assume that the default is mainly due to poor individual decisions, and such history tends to repeat itself. (Phillips, 2018) This implies that there are variables on each individual which could explain why some people have higher non-default probabilities than others.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

There exist adverse selection and moral hazard problems between borrowers and lenders. Besides, it is extremely costly to manually screen each borrower and to closely monitor the borrower whom the loan has been granted to. Therefore, there have been a lot of attempts in developing an automated credit scoring model which predicts non-default probabilities given individual attributes. One of the most widely used credit scoring models is logistic regression, where the dependent variable, i.e. non-default indicator variable, is a non-linear function of individual features. The logistic regression could provide a relatively simple explanation why the loan was denied. Due to its tractability and convenience in inference, the logistic regression is used in most local banks in the Czech and Slovak Republics. (Vojtek & Koèenda, 2006) Even though several advanced models outperform logistic regression, the logistic regression is still considered the industry standard for banks today. (Lessmann et al., 2015)

Logistic regression belongs to the class of supervised machine learning models. In order to collect enough data for training such a model, lenders have to face credit risk, and incur financial losses by collecting the labels of default borrowers. If a lender tries to avoid such an exposure by collecting just a few observations, the estimated model would have a high variance problem, called overfitting. In the opposite, if a lender collects too many observations, the improvement in the performance may not justify the financial losses from granting loan to default borrowers. To address the data acquisition cost in the model estimation process, reinforcement learning is proposed. In a reinforcement learning framework, an agent interacts with the environment several times to achieve some objective. (Sutton & Barto, 2018) In the credit scoring setting, an agent or lender chooses whom it should lend to, and feedbacks from such borrowers, either default or non-default, are returned to the agent to make this decision again in the next period.

The framework used in this study is the logistic bandit framework, which is a reinforcement learning framework that models non-default probabilities by the logistic regression. In this framework, there are a certain number of borrowers who apply for loans, and a reinforcement learning agent has to select a small number of borrowers whom credit would be granted to. A traditional credit scoring approach considers the model estimation process separately from the underwriting. This approach is identified as a greedy algorithm in the reinforcement learning literature, which is known to be suboptimal because it lacks an environment exploration. (Sutton & Barto, 2018) To enable an efficient learning, the agent has to explore by getting more information about the environment to improve model estimates while exploiting the information it has already gotten. One algorithm to address exploration in the greedy algorithm is a slight variation called epsilon-greedy. (Sutton & Barto, 2018) According to Russo et al. (2018), an even more efficient algorithm is Thompson sampling algorithm where the model estimate is sampled from posterior distribution. If an agent could directly sample an observation from the posterior distribution, Thompson sampling enables the agent to learn faster than epsilon-greedy. (Russo et al., 2018) However, since a direct sampling is not available in the logistic bandit framework, two algorithms to

approximate posterior sampling are used in this study, i.e. Laplace approximation and Langevin Monte Carlo Markov Chain (Langevin MCMC).

Many studies have performed comparisons on automated credit scoring systems in the form of machine learning classifiers using a fixed set of observations from datasets such as Vojtek and Koèenda (2006) and Lessmann et al. (2015). The limitation is that the model construction is considered separately from the underwriting process. Many studies have addressed the exploration problem in the logistic bandit framework such as Zhang et al. (2016), Dumitrascu et al. (2018) and Faury et al. (2020). The motivations of these studies are based on the recommender system and the online advertising, where an action is choosing one out of all possible products, and the set of products is fixed in every time step. However, in credit scoring, credit is often granted to multiple borrowers, and borrowers are not the same group of people in every time step. By allowing borrowers to be renewed from a population distribution, reinforcement learning would consider this as resampling an action set.

This study combines the credit scoring and the credit underwriting into a single sequential decision process, modelled by a logistic bandit framework. The unique characteristic of this logistic bandit model, which makes it different from traditional logistic bandit model, is that the actions are not fixed, but are randomly chosen from a population distribution in every time step. Different reinforcement learning algorithms are applied to the logistic bandit without and with sampled actions, and their performances are investigated. This study would provide a deeper understanding in reinforcement learning towards the logistic bandits, especially in credit scoring during credit underwriting processes.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

1.2 Objectives

To set a logistic bandit framework for credit scoring and credit underwriting process and to evaluate the performances of different reinforcement learning algorithms under this framework

1.3 Scope of Study

The algorithms investigated in this study include the greedy algorithm, the epsilon-greedy algorithm (the probability of exploration $\varepsilon = 0.05$), and Thompson Sampling, where the approximation algorithms are Laplace approximation and Langevin Monte Carlo Markov Chain. The performances of each algorithm are calculated using reward and regret in 250 time steps on 100 trials with the number of dimensions $p = 2, 10, 20$ under the following four scenarios.

1) An agent selects a single borrower, and the pool of borrowers is fixed. (Simple Setting)

2) An agent selects a single borrower, and the pool of borrowers is renewed under the same distribution after each time step.

3) An agent selects multiple (10) borrowers, and the pool of borrowers is fixed.

4) An agent selects multiple (10) borrowers, and the pool of borrowers is renewed under the same distribution after each time step. (Credit Scoring Setting)

1.4 Expected Benefits

A more efficient credit scoring and credit underwriting process is created under the reinforcement learning framework

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

CHAPTER II
RELATED WORKS

2.1 Logistic Regression

Logistic regression is a generalized linear model that attempts to model the probability that an observation falls into either of two classes using a linear combination of individual features, shown in Equation (1).

$$\Pr(y = 1|X = x) = \frac{1}{1 + \exp(-\beta_0 - \beta_1 x_1 - \cdots - \beta_p x_p)} \quad (1)$$

The model is fit via maximum likelihood estimation (MLE), where log likelihood of N observations $l(\beta)$ can be written as Equation (2).

$$l(\beta) = \sum_{i=1}^{N} \{y_i \beta^T x_i - \log(1 + e^{\beta^T x_i})\} \quad (2)$$

Setting the first-derivative of log-likelihood $l(\beta)$ with respect to β equal to zero, the result would be a system of non-linear equations in β. According to Hastie et al. (2009), to solve such a system, a common approach is to use the Newton-Raphson algorithm, which uses a Hessian matrix of log-likelihood with respect to β. The estimate of β, i.e. $\hat{\beta}$, is updated repeatedly using Newton step, and this algorithm is called iteratively reweighted least squares (IRLS). Since the log-likelihood of logistic regression is concave, the algorithm would typically converge to the optimal value, and $\hat{\beta} = 0$ is a good starting point for IRLS in this setting. (Hastie et al., 2009)

2.2 Bernoulli Bandit

This section presents the Bernoulli Bandit example from Russo et al. (2018). An agent is faced with three coins with probabilities of turning heads, i.e. p_1, p_2, p_3 which are unknown to the agent. In each time step, the agent selects one coin, and observes the tossing outcome in the next time step. The agent has to choose the coin with the highest probability turning head by repeatedly doing this several times. The objective is to maximize the cumulative rewards. The reinforcement learning

environment consists of an observation set, an action set and observation probabilities: $\mathcal{E} = (O, A, \rho)$ defined as follows.

1) Action set specifies which coin would be chosen: $A = \{1,2,3\}$

2) Observation set specifies all possible outcomes, either head (1) or tail (0): $O = \{0,1\}$

3) Observation probabilities specify the probability of a coin turning head given history and action: $\rho(1|H_t, A_t) = P(O_{t+1} = 1|H_t, A_t) = p_{A_t}$

To measure the performance of an algorithm, a simple method is to give a reward to the agent, and observe the reward as time step progresses. (Sutton & Barto, 2018) In Bernoulli bandit, a reward (R_{t+1}) is equal to an outcome (O_{t+1}). Over several trials, the result would be summarized using an average reward.

In an online learning, the performance is measured using regret. Regret is defined as the difference between expected reward of an optimal action and expected reward of the selected action. (Russo et al., 2018) In this setting, the regret from choosing an action A_t is displayed in Equation (3).

$$regret(t) = \max_{a \in A} p_a - p_{A_t} \quad (3)$$

If the agent chooses the optimal action, the regret would be equal to zero. Otherwise, regret would be positive. Similar to rewards, the result would be summarized using an average regret over simulation trials.

According to Sutton and Barto (2018), there is a trade-off between exploration and exploitation. Exploitation is when the current information is used to a full extent in order to select an action. Exploration is when an action with limited information is chosen in order to improve the estimate of how good that action actually is. Exploitation would maximize the reward in one step while exploration would compromise short-run reward in order to get large cumulative reward in the long-run. (Sutton & Barto, 2018) Since exploration and exploitation could not be done at the same time, Sutton and Barto (2018) concluded that there is a trade-off between these two.

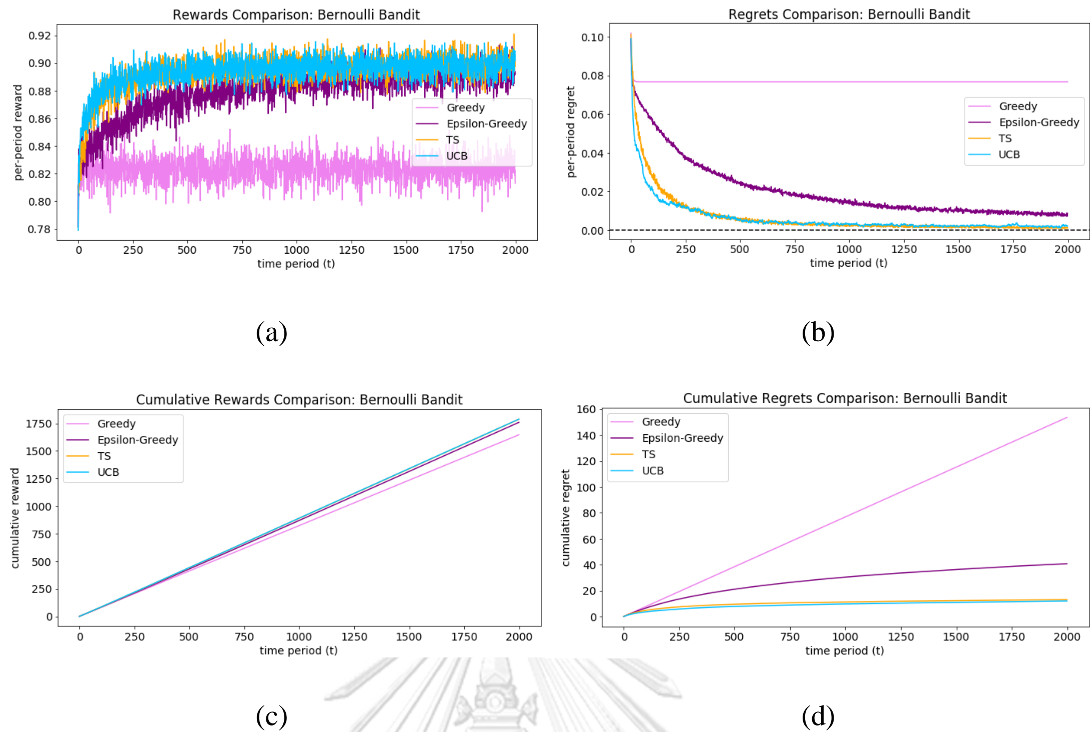
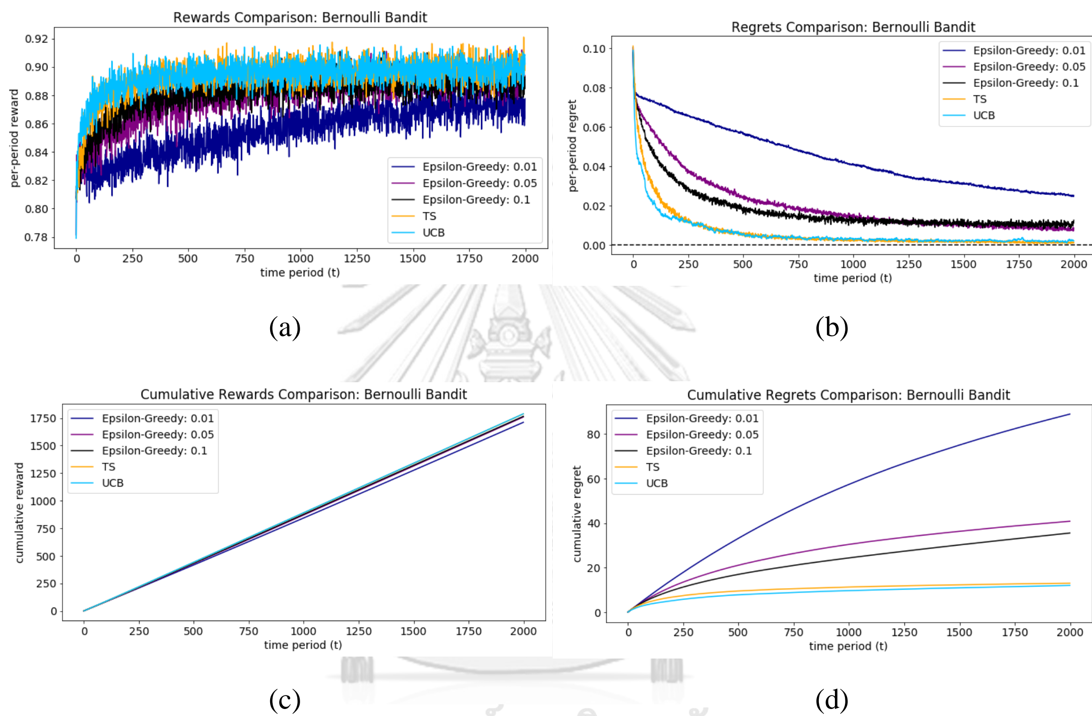


Figure 1. Performance comparisons on Bernoulli bandit

Four algorithms are used to select a coin, i.e. greedy, epsilon-greedy, Thompson Sampling (TS), and Upper Confidence Bound (UCB) illustrated in Figure 1. The greedy algorithm chooses the coin which maximizes the observed probability of turning head. According to Figure 1 (a) and (b), the greedy algorithm commits too early to just a few observations it had, and has not done any more explorations after a few time steps. This algorithm results in the lowest cumulative reward and the highest cumulative regret among all four algorithms in Figure 1 (c) and (d). A slight modification of greedy algorithm is the epsilon-greedy algorithm (Sutton & Barto, 2018), with a small probability to select one of three coins with an equal probability. The performance of this algorithm improves as the cumulative reward is higher and cumulative regret is lower in Figure 1 (c) and (d). However, the problem is that this algorithm does not efficiently explore the environment as it selects an action with an equal probability, regardless whether the action was actually a good action to explore. Two algorithms to solve such a problem are Thompson Sampling (TS) and Upper Confidence Bound (UCB). The TS chooses a coin by drawing the probability of turning head from a distribution, and selecting the coin with the highest such probability. The UCB

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

estimates the upper bound of the probability that each coin would turn head, and selects the coin with the highest such probability. In contrast to epsilon-greedy, both TS and UCB would spend efforts only on the coins where useful information could be obtained. (Russo et al., 2018) Therefore, both UCB and TS outperform epsilon-greedy due to higher cumulative rewards and lower cumulative regrets, shown in Figure 1 (c) and (d).



จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

Figure 2. Performances of epsilon-greedy ($\varepsilon = 0.01, 0.05, 0.1$) on Bernoulli bandit

Regarding the choices of epsilon (ε) in the epsilon-greedy algorithm on Bernoulli Bandit, the performances on the algorithm with $\varepsilon = 0.01, 0.05, 0.1$ are displayed in Figure 2. According to Figure 2 (a) and (b), small epsilon (0.01) results in the lowest per-period reward and the highest per-period regret. Even though large epsilon (0.1) triggers the lowest per-period regret during early time steps, its per-period regret becomes larger than the epsilon-greedy with epsilon of 0.05 in later time steps, shown in Figure 2 (b). Heavy exploration of epsilon 0.1 is beneficial at the beginning; however, once it is clear which coin yields the highest expected reward, the heavy exploration would become wasteful. Therefore, choosing epsilon (ε) in epsilon-greedy

algorithm involves a trade-off between exploration and exploitation. Still, with any levels of ε, the epsilon-greedy algorithm is inferior to both TS and UCB, which are efficient reinforcement learning algorithms in Figure 2 (d).

2.3 Logistic Bandit

In the logistic bandit framework, there are a certain number of possible actions that an agent can select. By observing the features associated with each action, an agent has to select the action that it estimates as the optimal one. In the next time step, the agent could observe the outcomes associated with the selected action. Such outcomes and features associated with the selected action would become the additional information that the agent could incorporate in order to select an action again.

Choosing a single action among all possible ones could be treated as selecting one of the coins in Bernoulli bandit. However, when the number of actions is very large, the problem will become difficult. (Zhang et al., 2016) Even Thompson sampling algorithm that is relatively good in exploration would suffer from having high regrets for a large number of time steps. Based on Zhang et al. (2016), one common approach to address this problem is using structural properties of reward function to build an efficient learning algorithm. The logistic bandit framework extends from Bernoulli bandit by incorporating generalization across actions. The generalization enables the information gained from selecting one action to be leveraged upon other actions. (Russo et al., 2018)

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

Similar to Logistic regression, the main feature of the logistic bandit framework is that an agent would get a binary feedback from environment. Since this main feature appears in a number of applications, in addition to Thompson Sampling (TS) from Russo et al. (2018), a number of algorithms have been developed to allow an efficient exploration by the agent; for example, OL^2M from Zhang et al. (2016), PG-TS from Dumitrascu et al. (2018), Logistic-UCB from Faury et al. (2020) and stochastic gradient descent with confidence ball strategy from Wang et al. (2017). However, these algorithms are based on Upper Confidence Bound strategy, except PG-TS which is based on Gibbs sampling, and they belong to different classes of algorithms from

Thompson Sampling in Russo et al. (2018). Therefore, they are not included in this study.

In this study, the logistic bandit is generalized to accommodate actions of selecting multiple borrowers and a renewed pool of borrowers. In reinforcement learning, selecting multiple borrowers means the number of actions is very large, and the renewed pool of borrowers means that actions are sampled.

2.4 Greedy Algorithm

In the setting with generalization across actions, Russo et al. (2018) has proposed the greedy algorithm that incorporates the generalization. According to Sutton and Barto (2018), the greedy algorithm will exploit all the information that the agent has collected to maximize the immediate reward without considering an exploration. This concept can be shown in Equation (4), where $Q_t(a)$ is the estimated value of an action a, and $N_t(a)$ is the number of time steps that the agent has chosen to take an action a.

$$A_t = \underset{a \in A}{\operatorname{argmax}}\, Q_t(a) = \underset{a \in A}{\operatorname{argmax}} \frac{R_1 + R_2 + \cdots + R_{N_t(a)}}{N_t(a)} \tag{4}$$

Hence, selecting k borrowers resulting in the highest estimated reward is equivalent to finding the estimate $\hat{\beta}$ that maximizes the posterior, given the history H_t, and using such estimated parameter $\hat{\beta}$ to select k borrowers with the highest estimated non-default probabilities. Then, the borrowing outcomes that the agent gets from the environment would be used to update the posterior. This is very similar to the greedy algorithm outlined in Russo et al. (2018).

The epsilon-greedy algorithm, a slight variation of greedy, allows an exploration. With probability of exploration ε, the algorithm could select k borrowers randomly from the pool of n borrowers where $k < n$. This is equivalent to choosing an action from all possible actions randomly with equal probabilities. (Sutton & Barto, 2018) With probability $1 - \varepsilon$, the algorithm estimates $\hat{\beta}$ that maximizes such posterior, and selects k borrowers with the highest estimated non-default probabilities.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

2.5 Thompson Sampling

According to Russo et al. (2018), Thompson sampling algorithm samples a parameter from a certain distribution, and the parameter is used to select the action that maximizes the estimated reward. Specifically, the agent starts with a prior distribution of the parameter $\hat{\beta}$. After $\hat{\beta}$ is sampled from this distribution, the estimated function is used to infer the probabilities of each individual falling into each class. Such estimates are used to select the action that would maximize the agent reward. In the next time step, the borrowing outcomes, together with features of selected borrowers, are used to update the posterior distribution of $\hat{\beta}$. (Russo et al., 2018) In other words, the estimated parameter $\hat{\beta}$ is updated using Bayes' rule. (Dumitrascu et al., 2018)

According to Russo et al. (2018), when posterior and prior are not conjugate distributions, an exact Bayesian inference would be difficult. To solve this problem, approximation methods are needed. In the logistic bandit setting, the functional form of logistic regression results in a computationally intractable posterior, which makes the implementation of Thompson sampling in this framework challenging. (Dumitrascu et al., 2018) In the following sections, two approximation algorithms are presented, i.e. Laplace approximation and Langevin MCMC.

2.6 Laplace Approximation

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

According to Gamerman and Lopes (2006), given the posterior $L(\beta)$ and its log transformation $l(\beta)$, Laplace approximation is based on a Taylor series expansion up until the second order around the presumably unique mode m, displayed in Equation (5), where $R(\beta)$ contains the third and above order, thus not considered in the approximation.

$$l(\beta) = l(m) + \left[\frac{\partial l(m)}{\partial \beta}\right]^T (\beta - m) - \frac{1}{2!}(\beta - m)^T \left[-\frac{\partial^2 \log L(m)}{\partial \beta \partial \beta^T}\right](\beta - m) + R(\beta) \qquad (5)$$

Since the posterior is often known only up to a proportionality constant, let $L^*(\beta) = kL(\beta)$. The Equation (5) can be rewritten as Equation (6), where the constant k is displayed in Equation (7), where $V = \left[-\frac{\partial^2 \ log\, L^*(m)}{\partial \beta \partial \beta^T}\right]^{-1}$, that is negative of the inverse Hessian matrix on $log\, L^*(\beta)$ calculated at the mode m.

$$L^*(\beta) \approx L^*(m) \exp\left\{-\frac{1}{2}(\beta - m)^T \left[-\frac{\partial^2 \log L^*(m)}{\partial\beta\partial\beta^T}\right](\beta - m)\right\} \quad (6)$$

$$k = L^*(m)(2\pi)^{d/2}|V|^{1/2} \quad (7)$$

Hence, β is approximately normally distributed at mode m and variance V, displayed in Equation (8).

$$\beta \dot{\sim} N(m, V) \quad (8)$$

A further approximation could be done by replacing m with $\hat{\beta}$ and replacing V with $I^{-1}(\hat{\beta})$ where $I(\cdot)$ is the observed Fisher information matrix, shown in Equation (9).

$$\beta \dot{\sim} N\left(\hat{\beta}, I^{-1}(\hat{\beta})\right) \quad (9)$$

For Laplace approximation to work well, the posterior L should be close in shape to a normal distribution as the approximation would ignore skewness and secondary mode. (Gamerman & Lopes, 2006) In the logistic bandit framework, Laplace approximation is a suitable algorithm since it could effectively fit a smooth density peaked around its mode. (Dumitrascu et al., 2018)

2.7 Metropolis Hasting

Hastings (1970) has proposed Metropolis Hasting as a sampling method based on Markov chain. Consider a distribution π simulated using a Markov chain. Define transition $p(x_i, x_j)$ that satisfies reversibility condition in Equation (10).

$$\pi(x_i)p(x_i, x_j) = \pi(x_j)p(x_j, x_i) \text{ for all } (x_i, x_j) \quad (10)$$

Assume the functional form of $p(x_i, x_j)$ follows Equation (11).

$$p(x_i, x_j) = q(x_i, x_j)\alpha(x_i, x_j) \text{ if } x_i \neq x_j \quad (11)$$

This implies the probability that the chain remains at x_i follows Equation (12).

$$p(x_i, x_i) = 1 - \int q(x_i, x_j)\alpha(x_i x_j)dx_j \quad (12)$$

In Gamerman and Lopes (2006), $q(x_i, x_j)$ is called a transition kernel, and $\alpha(x_i, x_j)$ is called an acceptance probability. Hastings (1970) defines the acceptance probability in Equation (13).

CHULALONGKORN UNIVERSITY

$$\alpha(x_i, x_j) = \begin{cases} 1 \; if \; \frac{\pi(x_j)q(x_j,x_i)}{\pi(x_i)q(x_i,x_j)} \geq 1 \\ \frac{\pi(x_j)q(x_j,x_i)}{\pi(x_i)q(x_i,x_j)} \; if \; \frac{\pi(x_j)q(x_j,x_i)}{\pi(x_i)q(x_i,x_j)} \leq 1 \end{cases} \tag{13}$$

Based on Gamerman and Lopes (2006), steps to implement Metropolis Hasting shall be outlined as follows.

Algorithm of Metropolis Hasting
1. Set $i = 0$, and an initial value of $x^{(0)}$
2. Generate a new value y from the transition kernel $q(x^{(i)}, \cdot)$
3. Calculate the acceptance probability $\alpha(x^{(i)}, y)$, from Equation (13).
4. The move is accepted with probability $\alpha(x^{(i)}, y)$. If the move is accepted, $x^{(i+1)} = y$. Otherwise, the value remains as $x^{(i+1)} = x^{(i)}$.
5. Increment i by 1, and repeat steps 2-4 for a number of times.

2.8 Langevin Monte Carlo Markov Chain (Langevin MCMC)

According to Karagulyan (2021), Langevin Monte Carlo sampling algorithm involves Euler-Maruyama discretization of the stochastic differential equation to Langevin diffusion. The discretization generates a proposal through the transition kernel. The proposal generation is displayed in Equation (14), where $\epsilon^{(i)}$ is normally distributed with zero mean and unit variance.

$$y = \beta^{(i)} + h\frac{\partial l(\beta^{(i)})}{\partial \beta} + \sqrt{2h}\epsilon^{(i)} \tag{14}$$

To correct the bias when discretizing, Metropolis-Hasting acceptance probability is used to either accept or reject the generated proposal in every number of iterations. (Karagulyan, 2021) The acceptance probability from Karagulyan (2021) is displayed in Equation (15).

$$\alpha(\beta^{(i)}, y) = \min\left(1, \frac{\exp\left(l(y) - \left(\frac{1}{4h}\right)\left(\left\|\beta^{(i)} - y - h\frac{\partial l(y)}{\partial \beta}\right\|_2^2\right)\right)}{\exp\left(l(\beta^{(i)}) - \left(\frac{1}{4h}\right)\left(\left\|y - \beta^{(i)} - h\frac{\partial l(\beta^{(i)})}{\partial \beta}\right\|_2^2\right)\right)}\right) \tag{15}$$

The implementation of Langevin MCMC is outlined in Karagulyan (2021) as follows.

Algorithm of Langevin Monte Carlo Markov Chain

1. Set $i = 0$, step size h, and an initial value of $\beta^{(0)}$
2. Generate a new value y from the proposal generation in Equation (14).
3. Calculate the acceptance probability $\alpha(\beta^{(i)}, y)$, from Equation (15).
4. The move is accepted with probability $\alpha(\beta^{(i)}, y)$. If the move is accepted, $\beta^{(i+1)} = y$. Otherwise, the value remains as $\beta^{(i+1)} = \beta^{(i)}$.
5. Increment i by 1, and repeat steps 2-4 for a number of times.

However, Karagulyan (2021) recommended using Metropolis adjustment step in the implementation since this algorithm is extremely sensitive to a fixed step-size h, and the chain could even be transient with a relatively large step size. Based on Russo et al. (2018), when a fixed step size is used in reinforcement learning, as time step progresses, the posterior density would become ill-conditioned, so an extremely small step size h is needed, causing the chain to converge very slowly. The approach suggested by Russo et al. (2018) is to adjust a step size using Hessian matrix, shown in Equation (16), where A is defined as the negative of inverse of the Hessian matrix of log likelihood calculated at $\beta^{(0)}$, $A = \left[-\frac{\partial^2 \log L^*(\beta^{(0)})}{\partial\beta\partial\beta^T}\right]^{-1}$.

$$y = \beta^{(i)} + hA\frac{\partial l(\beta^{(i)})}{\partial\beta} + \sqrt{2h}A^{1/2}\epsilon^{(i)} \tag{16}$$

Intuitively, whether a step size is adjusted or not, Euler-Maruyama discretization directs the move toward the proposal with high probability in L using its gradient direction. (Wang et al., 2017)

CHULALONGKORN UNIVERSITY

CHAPTER III
METHODOLOGY

This study models the credit scoring and underwriting into a logistic bandit, and conducts the simulation to evaluate reinforcement learning algorithms under different settings in the logistic bandit framework. The algorithms in this study include the greedy, the epsilon-greedy (the probability of exploration $\varepsilon = 0.05$) and Thompson Sampling, where the approximation algorithms employ Laplace approximation and Langevin MCMC. Performances of each algorithm are calculated using reward and regret in 250 time steps averaged over 100 simulation trials with the number of dimensions $p = 2, 10, 20$. The details of simulation are outlined in Section 3.1, the performance measures are indicated in Section 3.2, the implementations of each algorithm are specified in Section 3.3, and the diagram is presented in Section 3.4.

3.1 Logistic Bandit Framework and Simulation Method

In the traditional logistic bandit, the reinforcement learning agent chooses one of all available choices such as products, arms and borrowers. The agent attempts to select a single choice that yields the maximum expected reward. Furthermore, the action set available to the agent is fixed in every time step in the traditional logistic bandit. However, in credit scoring, the agent would grant loans to multiple borrowers, meaning that the agent would choose multiple borrowers per action. Also, the lender would face different borrowers in every time step, meaning that an action set is renewed. The main distinctions between these two settings are summarized in Table 1.

Table 1. The comparison between traditional logistic bandit and credit scoring

Features \ Settings	**Traditional Logistic Bandit**	**Credit Scoring**
Action Specification	Choosing a single borrower	Choosing multiple borrowers
Action Set	Fixed action set	Renewed action set

The logistic bandit environment has a ground truth parameter (β), a p-dimensional vector drawn from a standard normal distribution $N(0, I_p)$ with an intercept β_0 of 1.5. Once β is sampled from this distribution, the values β, β_0 does not change in each time step until the end of a simulation trial. In every time step, there are n borrowers available for loan to be granted ($n = 100$), where the feature of each borrower (X_i) is identically and independently distributed from a standard multivariate normal distribution with an identity covariance matrix. The parameter and feature vectors are independent. The probability of non-default on each borrower follows the logistic function with parameter (β) and borrower feature (X_i). The agent would observe a binary outcome $\{0,1\}$ based on the calculated probability of non-default. If the borrower whom agent granted loan to does not default, the agent would get a reward or $gain$. In case of default, the agent would be penalized with $loss$. In summary, the logistic bandit environment consists of a set of observations, a set of actions and observation probabilities, i.e. $\mathcal{E} = (O, A, \rho)$ outlined as follows.

1) Action set (A) specifies the borrowers whom loan would be granted, shown in Equation (17), where k is the number of borrowers that the agent could grant loan to.

$$A = \{a: a \subseteq \{1,2, \dots, n\}, |a| = k,\ k < n\} \tag{17}$$

2) Observation set (O) specifies whether each borrower is non-default (1) or default (0), shown in Equation (18).

$$O = \{0,1\}^k \tag{18}$$

3) Observation probabilities specify the probability of borrower i being non-default given history H_t and action A_t, shown in Equation (19), where $X_{t,i}$ is the feature of borrower i applying for a loan at time t.

$$\mathrm{P}\left(O_{t+1,i} = 1 | H_t, A_t, \mathcal{E}\right) = \frac{\exp(\beta^T X_{t,i})}{1+\exp(\beta^T X_{t,i})} \text{ for } i = 1, 2, \dots, k \text{ and } O_{t+1,i} \text{ are i.i.d.} \tag{19}$$

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

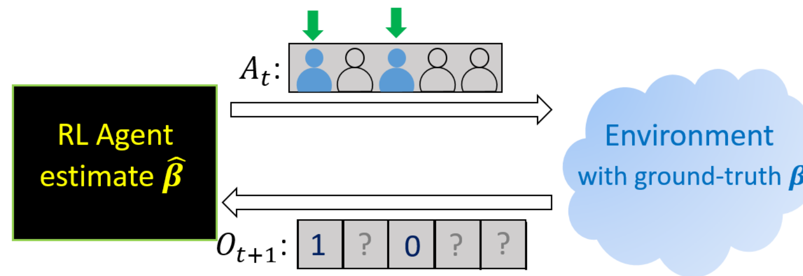


Figure 3. The interaction between agent and environment in logistic bandit framework

The interaction between agent and environment is illustrated in Figure 3. The environment is characterized by the ground-truth parameter β, which is used to generate an outcome $O_{t+1,i}$ according to the non-default probability calculated using Equation (19). The agent does not observe β, but could observe borrowers who were applying for loan ($X_{t,i}$). It would have to estimate $\hat{\beta}$ instead, which is used to infer the non-default probabilities of all borrowers, and to select k borrowers with the highest perceived non-default probabilities. Then, the borrowing outcomes of selected k borrowers would be returned to the agent as an additional information that agent would use to estimate $\hat{\beta}$ again.

The simulation is performed in 250 time steps on 100 simulation trials with the number of dimensions $p = 2, 10, 20$ under the following four scenarios.

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

1) An agent selects a single borrower, and the pool of borrowers is fixed. (Simple Setting)

2) An agent selects a single borrower, and the pool of borrowers is renewed under the same distribution after each time step.

3) An agent selects multiple (10) borrowers, and the pool of borrowers is fixed.

4) An agent selects multiple (10) borrowers, and the pool of borrowers is renewed under the same distribution after each time step. (Credit Scoring Setting)

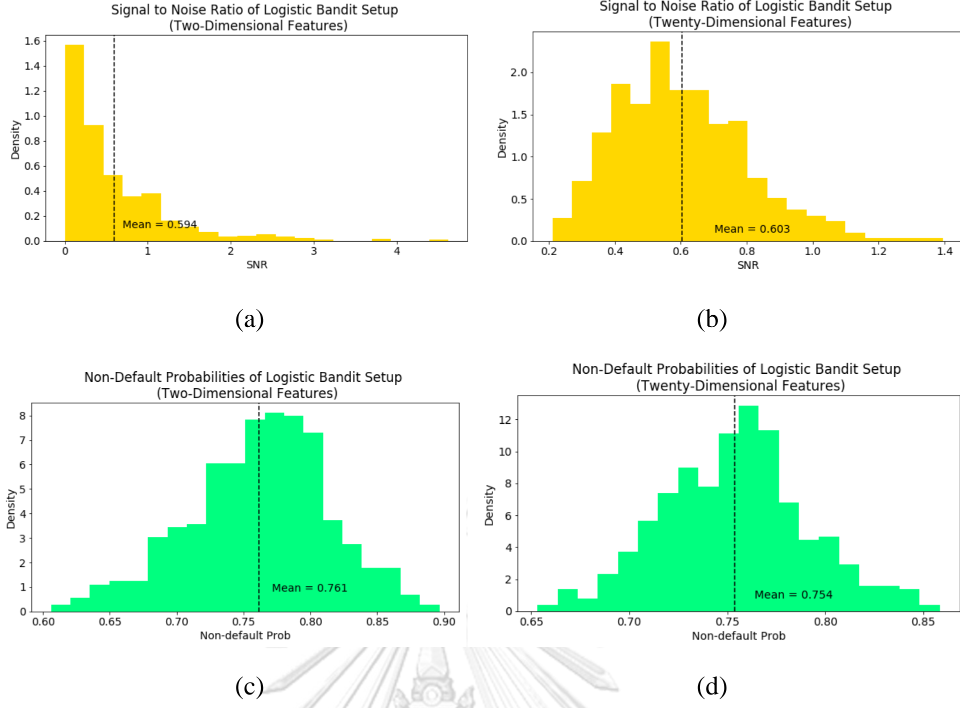


Figure 4. Histogram of sample average signal-to-noise ratio and sample average non-default probability under a small (2) and a large (20) number of features

This study evaluates the reinforcement learning algorithms not only on a small number of features ($p = 2$), but also on a large number of features ($p = 20$). To make different dimensional settings comparable, the adjustment coefficient $\sqrt{2/p}$ is multiplied with the borrower feature ($X_{t,i}$). The effects of the adjustment coefficient on signal-to-noise ratio and non-default probability are displayed in Figure 4. Signal-to-noise ratio is computed for the latent variable model with Logistic distribution using the formula in Soch and Allefeld (2018). After the adjustment, the signal-to-noise ratio is expected to be the same across different dimensions, and the sample means of signal-to-noise ratio would be similar, shown in Figure 4 (a) and (b). The sample means of non-default probabilities are also similar in both cases, shown in Figure 4 (c) and (d).

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

3.2 Performance Measures

The performance measures used in this study are rewards and regrets, following Sutton and Barto (2018) and Russo et al. (2018).

For the reward, if the borrower whom agent granted loan to does not default (the outcome is 1), the agent would get a reward of 0.2. In case of default (the outcome is 0), the agent would be penalized with a reward of -1. In each of 250 time steps, the reward or cumulative reward is summarized by averaging the (cumulative) reward on 100 simulation trials. According to Rosenberg et al. (2009) and Kneiding and Rosenberg (2008), in 2006, the average microfinance loan interest rate was 35% while the median interest rate was 26%. In Ethiopia and Sri Lanka, the interest rates were lower than 20%. (Kneiding & Rosenberg, 2008) Therefore, 20% interest rate is considered sustainable for microfinance loan. Without collaterals, a lender would lose the whole principal in case of default. If the number of borrowers whom the loan is granted (k) is greater than one, reward is the sum of rewards on each borrower divided by k.

According to Russo et al. (2018), regret is defined as the difference of maximum expected reward over all possible actions and expected reward of the taken action. The maximum expected reward is the expectation of reward based on k borrowers with the highest non-default probabilities, given $gain$ and $loss$. The expected reward of the taken action is the expectation of reward based on k borrowers that the agent has selected, given $gain$ and $loss$. This performance measure motivates the use of a simulation study because we knew the ground-truth expected reward of each action, which is required for the calculation of regret, and the regret is only available via the simulation.

CHULALONGKORN UNIVERSITY

3.3 Reinforcement Learning Algorithms

Because of generalization across actions, the agent can estimate $\hat{\beta}$ in order to estimate non-default probabilities of all borrowers in the pool. In this setting, an agent starts with prior of a standard normal distribution, indicated in Equation (20). Furthermore, the estimated parameter $\hat{\beta}$ is updated through the likelihood of logistic function, indicated in Equation (21).

$$f(\hat{\beta}) = \frac{1}{(2\pi)^{\frac{p+1}{2}}} \exp\left(-\frac{1}{2}\hat{\beta}^T\hat{\beta}\right) \tag{20}$$

$$f\left(\hat{\beta}; X_1, \dots, X_N, y_1, \dots, y_N\right) = \prod_{i=1}^{N} \left(\frac{\exp(\hat{\beta}^T X_i)}{1+\exp(\hat{\beta}^T X_i)}\right)^{y_i} \left(\frac{1}{1+\exp(\hat{\beta}^T X_i)}\right)^{1-y_i} \quad (21)$$

Based on Equation (20) and Equation (21), the posterior can be shown as Equation (22).

$$f\left(\hat{\beta} | X_1, \dots, X_N, y_1, \dots, y_N\right) \propto \quad (22)$$

$$\exp\left(-\frac{1}{2}\hat{\beta}^T\hat{\beta}\right) \prod_{i=1}^{N} \left(\frac{\exp(\hat{\beta}^T X_i)}{1+\exp(\hat{\beta}^T X_i)}\right)^{y_i} \left(\frac{1}{1+\exp(\hat{\beta}^T X_i)}\right)^{1-y_i}$$

For the greedy and the epsilon-greedy algorithm, the agent has to find $\hat{\beta}$ that maximizes the posterior in Equation (22). Finding such $\hat{\beta}$ is equivalent to fitting $\hat{\beta}$ to the logistic regression with L2 regularization. The implementation details of greedy and epsilon-greedy algorithm shall be specified in Algorithm 1, with the probability of exploration $(\varepsilon) = 0.05$. The choice of ε determines the trade-off between exploration and exploitation. According to Figure 2 (b) in Section 2.2 (Bernoulli bandit), the larger value of ε is, the better performance the algorithm achieved in the early time steps, but the algorithm would stay at the higher level of per-period regret in long run. In Bernoulli bandit setting, the epsilon-greedy algorithm is inferior to other algorithms with an efficient exploration, regardless of the choice of ε.

Algorithm 1: Greedy and Epsilon-Greedy

1: **Input:** $n, k, \varepsilon, time\ steps, gain, loss, renew$
2: **Output:** $rewards, regrets$
3: $X \leftarrow \emptyset, Y \leftarrow \emptyset, rewards \leftarrow [\ \ ], regrets \leftarrow [\ \ ], warm\ start \leftarrow True$
4: Initialize the time-step counter $t \leftarrow 1$
5: while ($t < time\ steps$):
6:     Sample u from $Unif(0,1)$
7:     If ($warm\ start$ or $u \leq \varepsilon$): choose k borrowers from n borrowers randomly
8:     Else: find $\hat{\beta}$ that maximizes the posterior in Equation (22), and use that $\hat{\beta}$ to select k borrowers with the estimated highest non-default probabilities
9:     Append $rewards, regrets$ using the current period reward and regret calculated using $gain, loss$
10:     Update the history: $X \leftarrow X \cup \{X_{1,t}, \dots, X_{k,t}\}, Y \leftarrow Y \cup \{y_{1,t}, \dots, y_{k,t}\}$
11:     If (Y contains both 0 and 1): $warm\ start \leftarrow False$
12:     If ($renew$): Resample p-dimensional features of n borrowers
13:     $t \leftarrow t + 1$
14: end

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

For Thompson sampling, the algorithm samples an estimated parameter $\hat{\beta}$ from the posterior distribution, and its value is used to select the action that maximizes the estimated expected reward. Then, the borrowing outcomes and individual features of k selected borrowers are used to update the posterior distribution of $\hat{\beta}$ via Bayes' rule. (Dumitrascu et al., 2018) The implementation details of Thompson sampling shall be specified in Algorithm 2.

Algorithm 2: Thompson Sampling

1: **Input:** $n, k, time\ steps, gain, loss, renew$
2: **Output:** $rewards, regrets$
3: $X \leftarrow \emptyset, Y \leftarrow \emptyset, rewards \leftarrow [\ \ ], regrets \leftarrow [\ \ ], warm\ start \leftarrow True$
4: Initialize the time-step counter $t \leftarrow 1$
5: while ($t < time\ steps$):
6: If ($warm\ start$): choose k borrowers from n borrowers randomly
7: Else: Draw $\hat{\beta}$ from the posterior distribution using an approximation algorithm, and use that $\hat{\beta}$ to select k borrowers with the estimated highest non-default probabilities
8: Append $rewards, regrets$ using the current period reward and regret calculated using $gain, loss$
9: Update the history: $X \leftarrow X \cup \{X_{1,t}, \dots, X_{k,t}\}, Y \leftarrow Y \cup \{y_{1,t}, \dots, y_{k,t}\}$
10: If (Y contains both 0 and 1): $warm\ start \leftarrow False$
11: If ($renew$): Resample p-dimensional features of n borrowers
12: $t \leftarrow t + 1$
13: end

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

However, if the posterior distribution is not a conjugate of the prior distribution, an exact Bayesian inference would be difficult. (Russo et al., 2018) In the logistic bandit framework, the functional form of logistic regression results in computationally intractable posterior, which makes the application of Thompson sampling in this framework challenging. Dumitrascu et al. (2018) To address this problem, an approximation method will be used. This study uses Laplace approximation and Langevin Monte Carlo Markov Chain algorithm, which are both popular approximation algorithms for Thompson sampling.

Laplace approximation algorithm follows Gamerman and Lopes (2006), where the estimated parameter $\hat{\beta}$ is fit using the logistic regression with L2 regularization,

similar to Algorithm 1, and covariance matrix is negative of the inverse Hessian matrix on $log\, L^{*}(\beta)$ $(L^{*}(\beta) = kL(\beta))$ calculated at the estimated $\hat{\beta}$. The implementation details of Laplace approximation shall be specified in Algorithm 3.

Algorithm 3: Laplace Approximation

1: **Input:** X, Y
2: **Output:** $\hat{\beta}$
3: Let $mode \leftarrow \hat{\beta}$ that maximizes the posterior probability in Equation (22), given X, Y
4: Let $cov \leftarrow$ the inverse of Fisher information matrix with respect to the posterior distribution calculated at $mode$, given X, Y
5: Sample $\hat{\beta}$ from the normal distribution: $\hat{\beta} \sim N(mode, cov)$

Langevin Monte Carlo Markov Chain consists of a proposal generation and an acceptance probability. The proposal generation follows the algorithm with an adaptive step size from Russo et al. (2018), which is specified in Equation (23), where $\epsilon^{(i)}$ is normally distributed with zero mean and unit variance, $l(\beta)$ is logarithm of posterior, and the adaptive step-size matrix A is indicated in Equation (24).

$$y = \beta^{(i)} + hA\frac{\partial l(\beta^{(i)})}{\partial \beta} + \sqrt{2h}A^{1/2}\epsilon^{(i)} \tag{23}$$

$$A = \left[-\frac{\partial^2 l(\beta^{(0)})}{\partial\beta\partial\beta^T}\right]^{-1} \tag{24}$$

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

For the acceptance probability, Metropolis-Hasting acceptance probability from Hastings (1970) and Karagulyan (2021) is used to either accept or reject the generated proposal in every iteration. By incorporating the adaptive step-size, the acceptance probability can be derived as Equation (25).

$$\alpha(\beta^{(i)}, y) = \min\left(1, \frac{\exp(upper)}{\exp(lower)}\right) \tag{25}$$

$$upper = l(y) - \left(\frac{1}{4h}\right)\left(\beta^{(i)} - y - hA\frac{\partial l(y)}{\partial \beta}\right)' A^{-1}\left(\beta^{(i)} - y - hA\frac{\partial l(y)}{\partial \beta}\right)$$

$$lower = l(\beta^{(i)}) - \left(\frac{1}{4h}\right)\left(y - \beta^{(i)} - hA\frac{\partial l(\beta^{(i)})}{\partial \beta}\right)' A^{-1}\left(y - \beta^{(i)} - hA\frac{\partial l(\beta^{(i)})}{\partial \beta}\right)$$

The implementation details of Langevin Monte Carlo Markov Chain shall be specified in Algorithm 4 with the step size h is 2, and the $num\ iters$ are 100.

Algorithm 4: Langevin Monte Carlo Markov Chain

1: **Input:** $X, Y, h, num\ iters$
2: **Output:** $\hat{\beta}$
3: Initialize $\beta^{(0)} \leftarrow \hat{\beta}$ that maximizes the posterior probability in Equation (22) given X, Y
4: Initialize the counter $i \leftarrow 1$
5: while ($i < num\ iters$):
6: Generate a proposal y using the proposal generation in Equations (23-24)
7: Calculate the Metropolis-Hasting acceptance probability $\alpha(\beta^{(i)}, y)$ in Equation (25)
8: Sample u from $Unif(0,1)$
9: If ($u \leq \alpha(\beta^{(i)}, y)$): assign $\beta^{(i+1)} \leftarrow y$
10: Else: the chain does not move $\beta^{(i+1)} \leftarrow \beta^{(i)}$
11: $i \leftarrow i + 1$
12: end
13: Retrieve the last value as the estimated parameter: $\hat{\beta} \leftarrow \beta^{(i)}$

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

3.4 Algorithm Flowchart

The algorithm flowchart of modelling credit scoring and underwriting, and evaluating reinforcement learning algorithms under the simulation study is outlined in Figure 5.

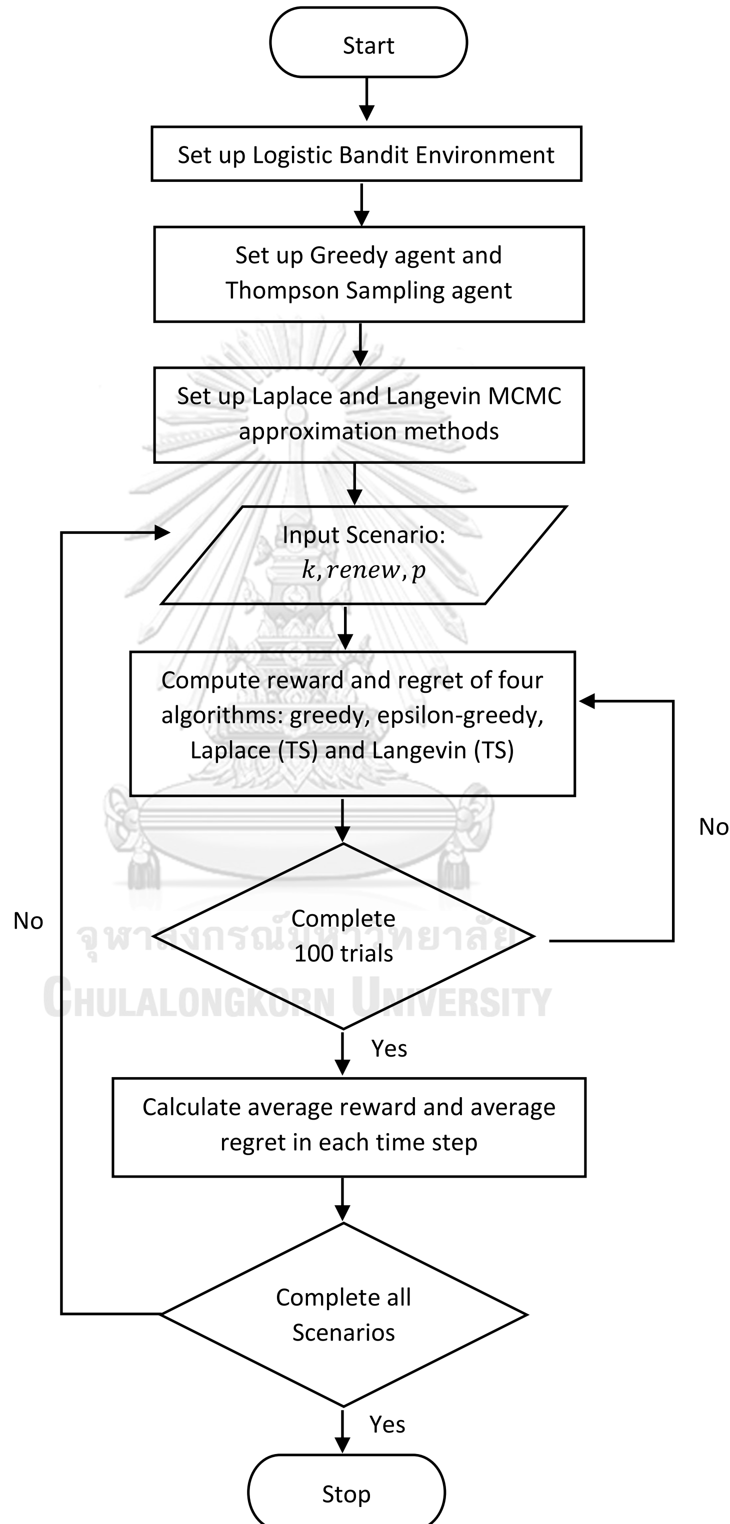


Figure 5. Algorithm flowchart

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

CHAPTER IV
RESULTS

The purpose of this study is to model the credit scoring and underwriting into a logistic bandit, and evaluate the performances of four reinforcement learning algorithms based on the logistic bandit framework. The algorithms used in this study include the greedy, the epsilon-greedy, Thompson sampling with Laplace approximation and Thompson sampling with Langevin Monte Carlo Markov Chain. The greedy can be deemed as the traditional approach for credit scoring and underwriting while Thompson sampling can be deemed as a reinforcement learning algorithm that includes an efficient exploration.

Two performance measures used to evaluate these algorithms are rewards and regrets over 250 time steps, averaged on 100 simulation trials under the number of dimensions $p = 2, 10, 20$. For rewards, if the borrower did not default on loan, the agent would get a reward of 0.2. Otherwise, the agent would get a reward of -1. If k borrowers are granted loans, the reward would be the sum of reward on each of the borrower divided by k. The regret is the difference between the maximum expected reward over all possible actions, and the expected reward of the selected action.

Two performance measures on four reinforcement learning algorithms are investigated under the following scenarios.

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

1) Scenario 1: The agent selects a single borrower while the pool of borrowers is fixed. This is the traditional logistic bandit usually found in literatures discussing the logistic bandit framework.

2) Scenario 2: The agent selects a single borrower while the pool of borrowers is renewed after each time step. In credit scoring, lenders would actually find new customers applying for a loan.

3) Scenario 3: The agent selects multiple (10) borrowers while the pool of borrowers is fixed. In microfinance or consumer loans, lenders would give small amounts of loans to many customers at once.

4) Scenario 4: The agent selects multiple (10) borrowers while the pool of borrowers is renewed after each time step. This is the credit scoring setting that incorporates two modifications which capture the main characteristics of the credit scoring and underwriting processes.

4.1 Small Number of Features ($p = 2$)

This section shows the performances of each reinforcement learning algorithm when borrower characteristics are captured by a small number of features ($p = 2$) under the following four scenarios.

1) Scenario 1: Selecting a Single Borrower without Renewal

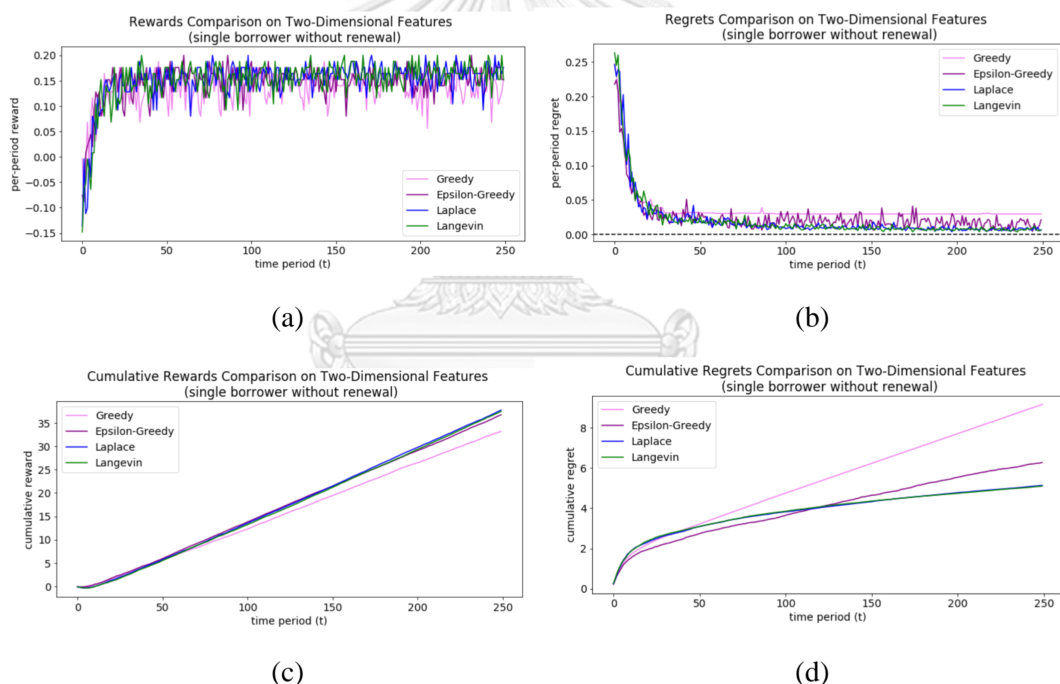


Figure 6. Performance comparisons on two-dimensional features (single borrower without renewal)

Figure 6 illustrates the performance of each algorithm when the borrower characteristics are captured by a two-dimensional vector, where the setting is that a single borrower is selected per action while the pool of borrowers is fixed. The greedy algorithm chooses the action based on the parameter which maximizes the posterior, which is based on only a few observations in early time steps. According to Figure 6

(c) and (d), the greedy performs the worst due to its lowest cumulative reward and its highest cumulative regret as the algorithm commits too early to an inferior action. The epsilon-greedy algorithm modifies the greedy algorithm by incorporating an exploration: with a small chance, the algorithm selects one of all possible actions with equal probabilities. The epsilon-greedy algorithm performs better than the greedy algorithm; however, its cumulative regret increases in a linear fashion, shown in Figure 6 (d). Thompson sampling algorithms conduct an efficient exploration by exploring an action with limited information. With either Laplace approximation or Langevin MCMC, Thompson sampling algorithms perform quite well by achieving high cumulative rewards and low cumulative regrets, shown in Figure 6 (c) and (d). Based on Figure 6 (d), the cumulative regrets of Thompson sampling are high in early time steps due to heavy exploration whereas the cumulative regrets become lower in later time steps because the agent has enough information for exploitation.

2) Scenario 2: Selecting a Single Borrower with Renewal

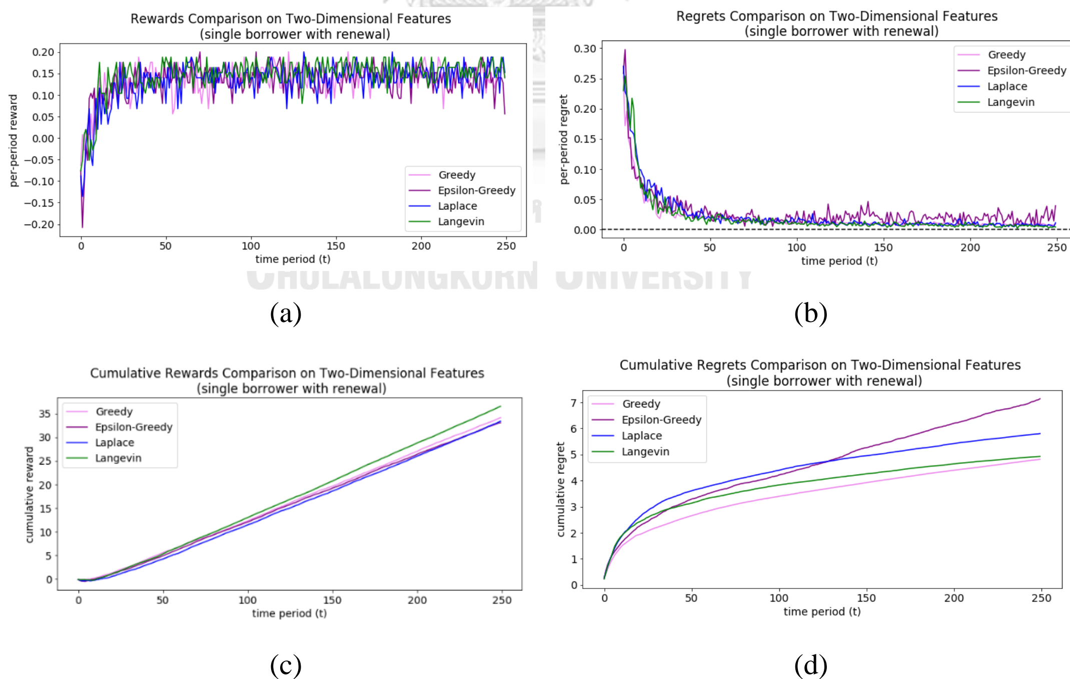
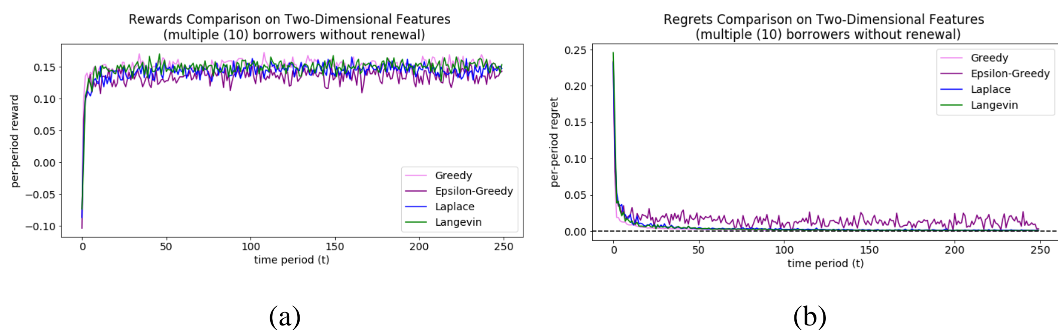


Figure 7. Performance comparisons on two-dimensional features (single borrower with renewal)

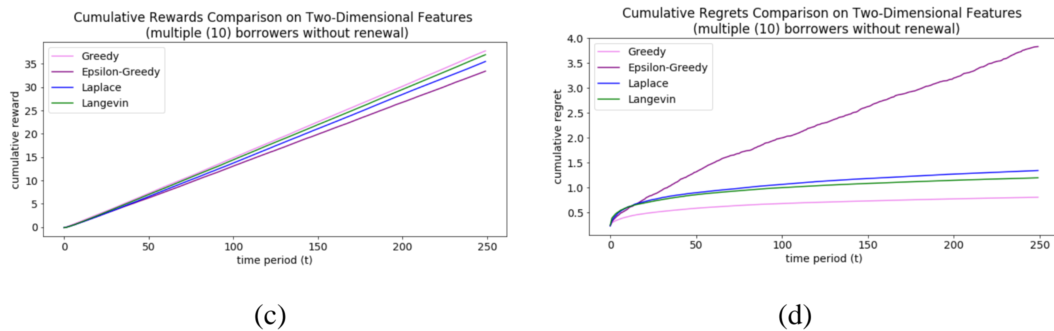
Figure 7 illustrates the performance of each algorithm when the borrower characteristics are captured by a two-dimensional vector, where the setting is that a single borrower is selected per action while the pool of borrowers is renewed. The epsilon-greedy performs the worst due to its lowest cumulative reward and highest cumulative regret, shown in Figure 7 (c) and (d). The algorithm selects an action randomly with a small chance in every time step, causing the linear increase in cumulative regret, shown in Figure 7 (d). In Figure 7 (c) and (d), Thompson sampling algorithms with either approximation method perform better than the epsilon-greedy because they conduct an efficient exploration, i.e. heavy exploration in beginning time steps, and heavy exploitation in later time steps. Unlike Scenario 1 in Figure 6, the greedy algorithm achieves higher cumulative rewards than Thompson sampling with Laplace approximation, and the greedy algorithm achieves lower cumulative regrets than Thompson sampling algorithms with either approximation method, shown in Figure 7 (c) and (d). The result implies that the renewal of borrowers allows the greedy algorithm to perform quite well as the algorithm would not commit too early to an inferior action. In other words, there is no single inferior borrower for the greedy algorithm to commit to since the pool of borrowers is renewed in every time step. In a sense, the renewal of the borrower pool enables a kind of exploration into the greedy algorithm that otherwise would not explore the action space.

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

3) Scenario 3: Selecting Multiple (10) Borrowers without Renewal



(a) (b)

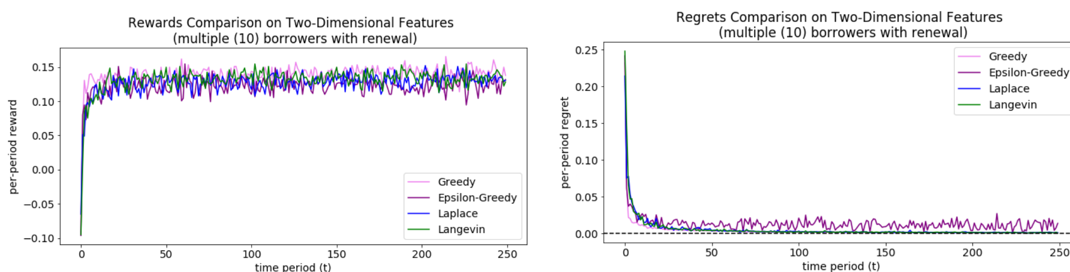


(c) (d)

Figure 8. Performance comparisons on two-dimensional features (multiple borrowers without renewal)

Figure 8 illustrates the performance of each algorithm when the borrower characteristics are captured by a two-dimensional vector, where the setting is that multiple (10) borrowers are selected per action while the pool of borrowers is fixed. In Figure 8 (d), the epsilon-greedy performs the worst because of the linear increase in cumulative regret, similar to Figure 7 (d). Based on Figure 8 (c) and (d), Thompson sampling algorithms with either approximation method perform significantly better than the epsilon-greedy in terms of the cumulative rewards and the cumulative regrets because of efficient exploration. Unlike Scenario 1 in Figure 6, the greedy algorithm achieves higher cumulative rewards and lower cumulative regrets than Thompson sampling algorithms with either approximation method, shown in Figure 8 (c) and (d). By choosing multiple borrowers per action, the agent has enough information to choose the parameter that results in borrowers with the highest non-default probabilities without explicitly exploring the action space as Thompson sampling algorithms, shown in Figure 8 (a) and (b).

4) Scenario 4: Selecting Multiple (10) Borrowers with Renewal



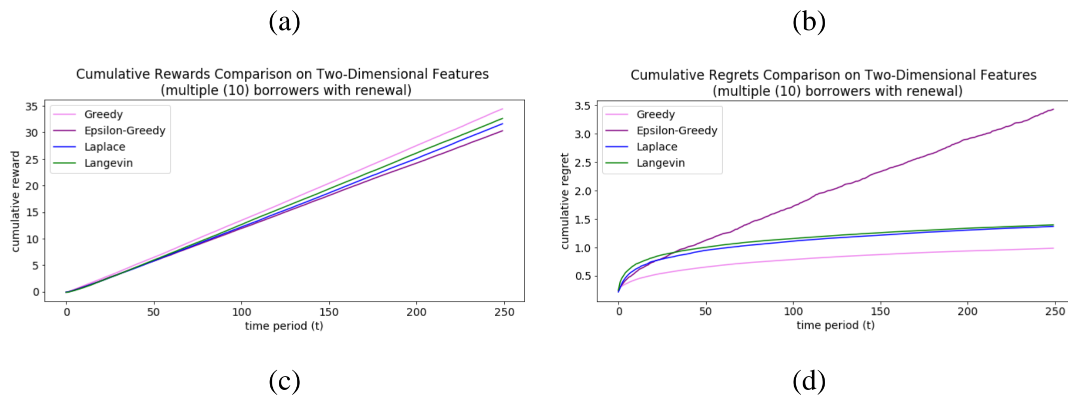


Figure 9. Performance comparisons on two-dimensional features (multiple borrowers with renewal)

Figure 9 illustrates the performance of each algorithm when the borrower characteristics are captured by a two-dimensional vector, where the setting is that multiple (10) borrowers are selected per action while the pool of borrowers is renewed at every time step. In Figure 9 (c) and (d), the epsilon-greedy algorithm performs the worst due to the linear increase in the cumulative regret, similar to Figure 7 (d) and Figure 8 (d). Thompson sampling algorithms with either approximation method performs an efficient exploration, resulting in higher cumulative rewards and lower cumulative regrets than the epsilon-greedy, shown in Figure 9 (c) and (d). Unlike Scenario 1 in Figure 6, the greedy algorithm could outperform Thompson sampling algorithms in terms of cumulative rewards and cumulative regrets, shown in Figure 9 (c) and (d). In this scenario, the pool of borrowers is renewed, enabling a kind of exploration in the greedy algorithm in Scenario 2 whereas the agent could provide loans to multiple borrowers per action, resulting in enough information to select borrowers by the agent in Scenario 3. These two modifications allow the greedy algorithm to outperform Thompson sampling algorithms.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

4.2 Medium Number of Features ($p = 10$)

This section shows the performances of each reinforcement learning algorithm when borrower characteristics are captured by a medium number of features ($p = 10$) under the following four scenarios.

1) Scenario 1: Selecting a Single Borrower without Renewal

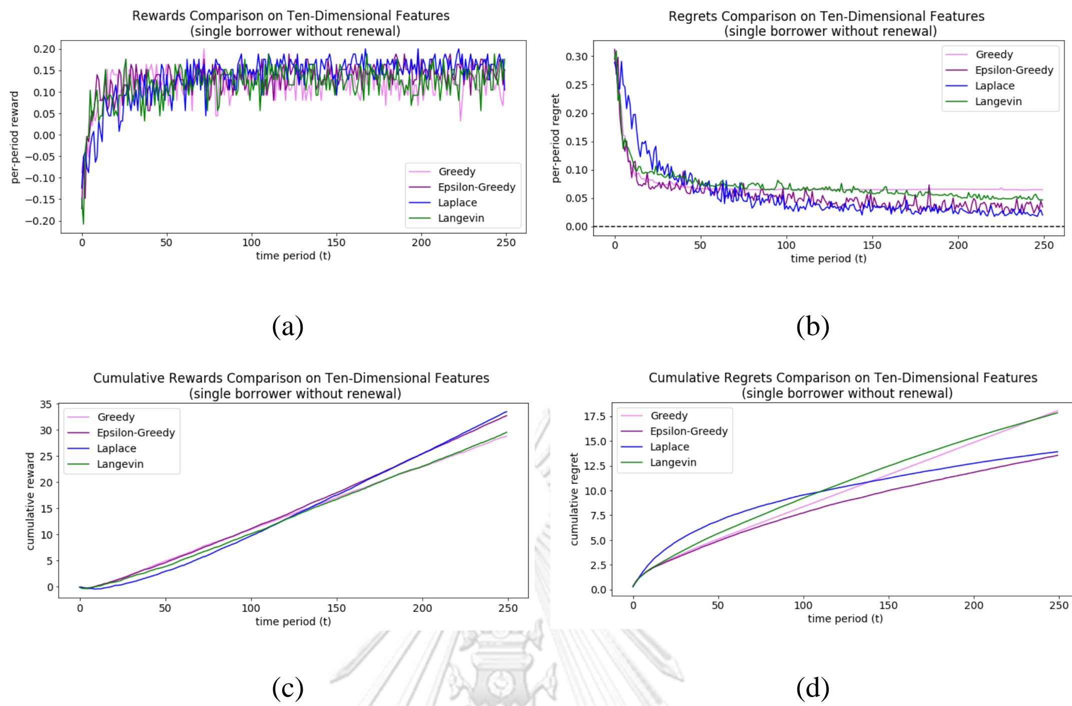


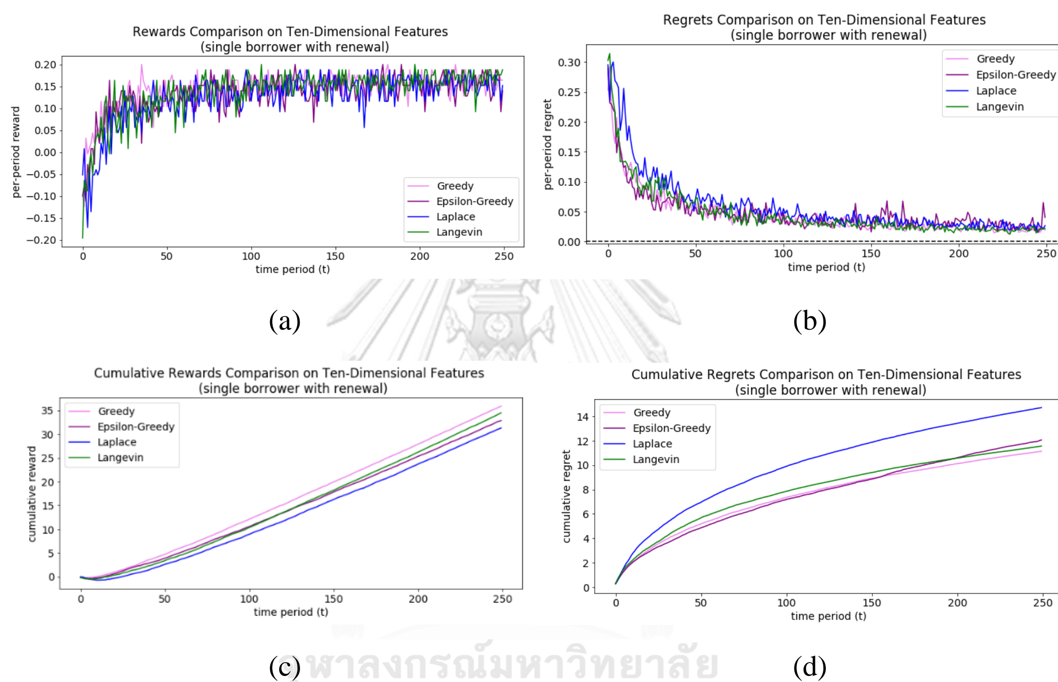
Figure 10. Performance comparisons on ten-dimensional features (single borrower without renewal)

Figure 10 illustrates the performance of each algorithm when the borrower characteristics are captured by a ten-dimensional vector, where the setting is that a single borrower is selected per action while the pool of borrowers is fixed. Overall, by increasing the number of features used to capture borrower characteristics, per-period regrets in later time steps of each algorithm are higher, implying that this setting is more difficult for reinforcement learning algorithms, shown in Figure 10 (b). In Figure 10 (b), per-period regret of the greedy algorithm is still higher than other algorithms, similar to Scenario 1 in Figure 6. Thompson sampling algorithm with Langevin MCMC performs a heavy exploration before reaching lower per-period regret than the greedy algorithm, shown in Figure 10 (b). The number of time steps the algorithm needs to perform heavy exploration is higher than Scenario 1 in Figure 6 as the number of features capturing borrower characteristics increases from two to ten. Thompson sampling with Laplace approximation performs heavy exploration in fewer number of time steps than Thompson sampling with Langevin MCMC, resulting in better performances in cumulative rewards and cumulative regrets, shown in Figure 10 (c)

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

and (d). The epsilon-greedy algorithm does not conduct heavy exploration, but its per-period regret is greater than Thompson sampling with Laplace approximation in later time steps, shown in Figure 10 (b). This implies that the epsilon-greedy does not conduct an efficient exploration, resulting in the linear increase in cumulative regret in Figure 10 (d).

2) Scenario 2: Selecting a Single Borrower with Renewal



จุฬาลงกรณ์มหาวิทยาลัย

Figure 11. Performance comparisons on ten-dimensional features (single borrower with renewal)

Figure 11 illustrates the performance of each algorithm when the borrower characteristics are captured by a ten-dimensional vector, where the setting is that a single borrower is selected per action while the pool of borrowers is renewed in every time step. As the number of borrower features increases from two to ten, the setting becomes more difficult, and Thompson sampling with Laplace approximation needs to perform heavy exploration in a number of time steps before reaching same level of per-period regret as other algorithms, shown in Figure 11 (c). This results in the lowest cumulative regret and the highest cumulative reward of this algorithm in Figure 11 (d). As Thompson sampling with Langevin MCMC does not perform such the heavy

exploration as Thompson sampling with Laplace approximation, shown in Figure 11 (b), the algorithm reaches higher cumulative reward and lower cumulative regret, shown in Figure 11 (c) and (d). The epsilon-greedy achieves low cumulative regret in early time step due to a small chance of exploration in every time step, and its cumulative regret increases linearly, resulting in higher cumulative regrets than Thompson sampling with Langevin MCMC. Similar to Scenario 2 in Figure 7, the renewal of borrower pool allows the greedy algorithm to outperform other algorithms in terms of cumulative rewards and cumulative regrets, shown in Figure 11 (c) and (d).

3) Scenario 3: Selecting Multiple (10) Borrowers without Renewal

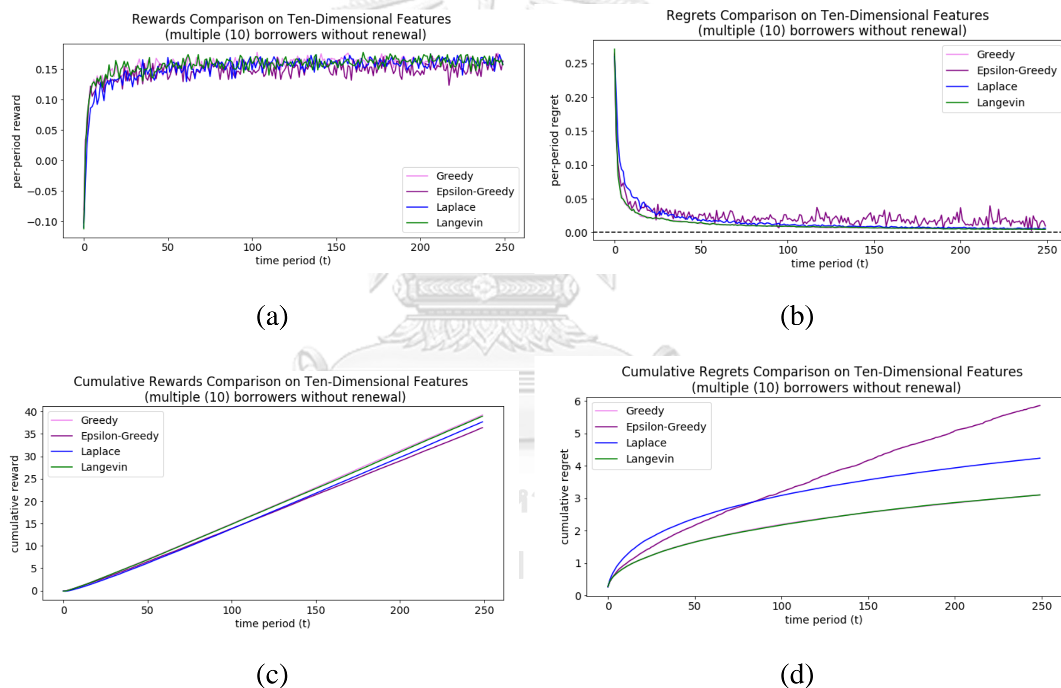
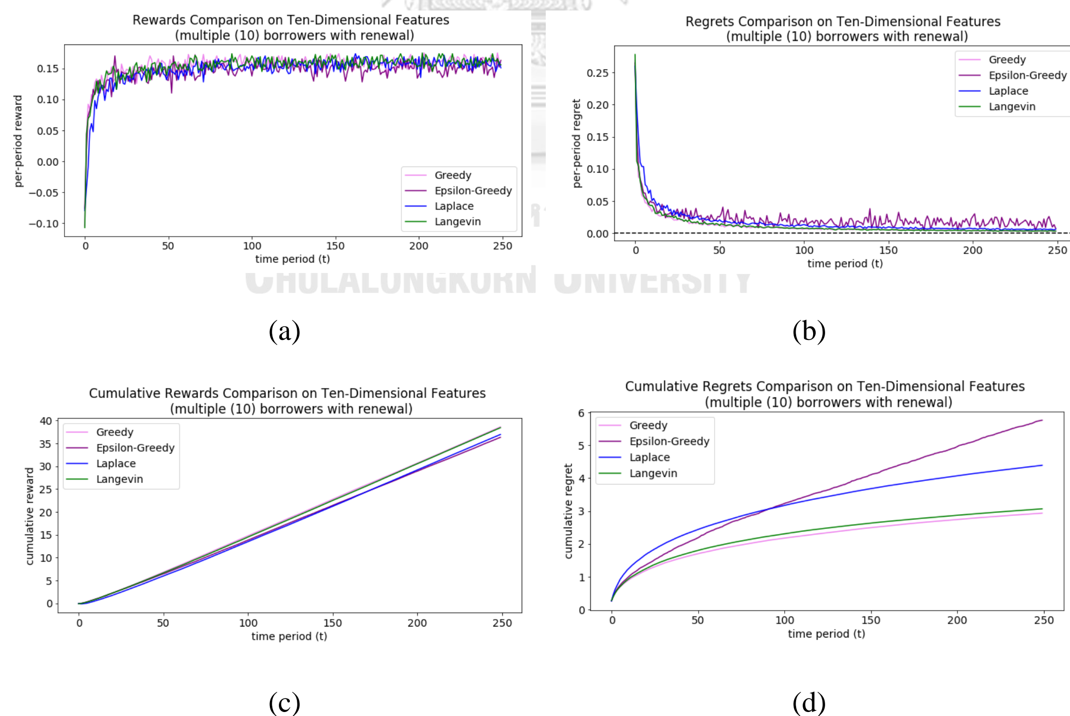


Figure 12. Performance comparisons on ten-dimensional features (multiple borrowers without renewal)

Figure 12 illustrates the performance of each algorithm when the borrower characteristics are captured by a ten-dimensional vector, where the setting is that multiple (10) borrowers are selected per action while the pool of borrowers is fixed. In Figure 12 (b) and (d), the epsilon-greedy results in the highest cumulative regret as the cumulative regret increases linearly due to a small chance of an exploration in every

time step. Similar to Scenario 2 in Figure 11, Thompson sampling algorithm with Laplace approximation needs to perform quite a heavy exploration in beginning time steps, indicated by high per-period regret in Figure 12 (c), but it managed to achieve lower cumulative regret than the epsilon-greedy in later time steps, shown in Figure 12 (d). Because Thompson sampling with Langevin MCMC does not perform such the heavy exploration as Thompson sampling with Laplace approximation, the algorithm manages to get higher cumulative reward and lower cumulative regret, shown in Figure 12 (c) and (d). The greedy algorithm achieves the performance close to Thompson sampling with Langevin MCMC, shown in Figure 12 (c) and (d). Unlike Scenario 3 in Figure 8, as the number of features to capture borrower characteristics is greater, the greedy algorithm would face some difficulties in utilizing the same number of borrowers in each time step in order to find parameters that could maximize cumulative rewards.

4) Scenario 4: Selecting Multiple (10) Borrowers with Renewal



(a) (b)

(c) (d)

Figure 13. Performance comparisons on ten-dimensional features (multiple borrowers with renewal)

Figure 13 illustrates the performance of each algorithm when the borrower characteristics are captured by a ten-dimensional vector, where the setting is that multiple (10) borrowers are selected per action while the pool of borrowers is renewed in every time step. In Figure 13 (b) and (d), the cumulative regret of the epsilon-greedy increases linearly, resulting in the worst performance in later time steps. Similar to Scenario 2 in Figure 11 and Scenario 3 in Figure 12, Thompson sampling with Laplace approximation results in higher per-period regrets in beginning time steps and lower cumulative regrets in later time steps due to efficient exploration, shown in Figure 13 (b) and (d). Compared with Scenario 4 in Figure 9, as the number of borrower features is greater, Thompson sampling with Laplace approximation needs to perform a heavier exploration in a number of time steps before settling down to lower level of per-period regret. Thompson sampling with Langevin MCMC does not conduct quite heavy exploration as Thompson sampling with Laplace approximation, resulting in lower cumulative regrets, shown in Figure 13 (b) and (d). As Scenario 4 incorporates two modifications which allows the greedy algorithm to perform better, the greedy algorithm achieves the lowest cumulative regret in Figure 13 (d). Unlike Scenario 4 in Figure 9, as the number of borrower features is greater, the gap in cumulative regrets between the greedy and Thompson sampling with Langevin MCMC narrows down considerably.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

4.3 Large Number of Features ($p = 20$)

This section shows the performances of each reinforcement learning algorithm when borrower characteristics are captured by a large number of features ($p = 20$) under the following four scenarios.

1) Scenario 1: Selecting a Single Borrower without Renewal

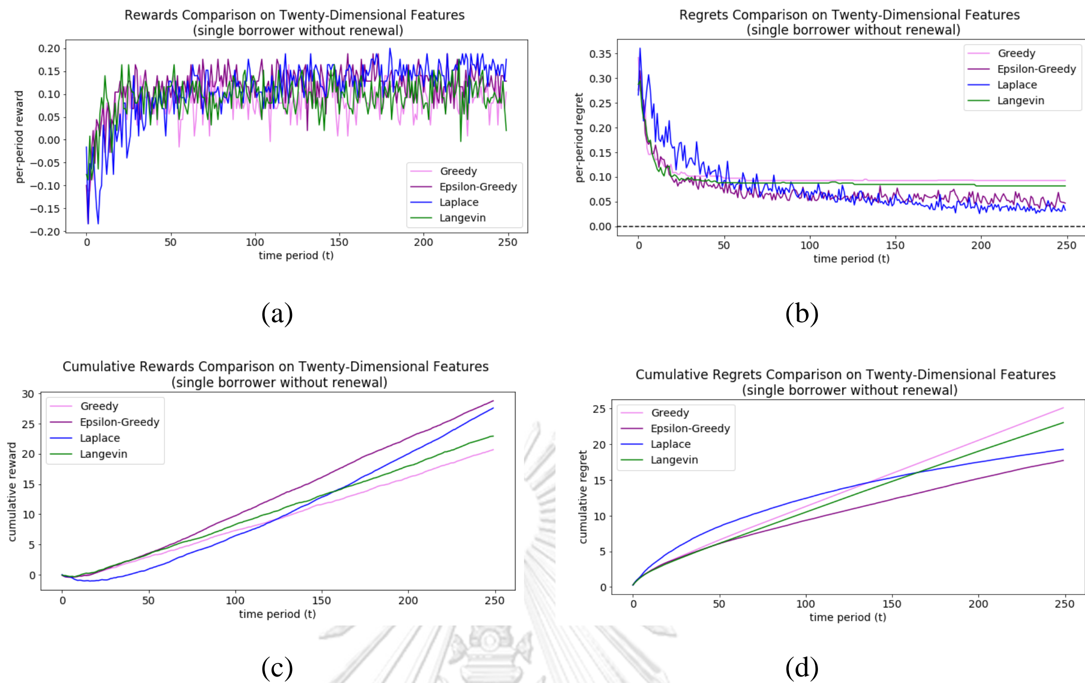


Figure 14. Performance comparisons on twenty-dimensional features (single borrower without renewal)

Figure 14 illustrates the performance of each algorithm when the borrower characteristics are captured by a twenty-dimensional vector, where the setting is that a single borrower is selected per action while the pool of borrowers is fixed. Overall, by increasing the number of borrower features from ten to twenty, per-period regrets in later time steps of each algorithm are higher, indicating that the problem is more difficult for reinforcement learning algorithms. The greedy algorithm results in the lowest cumulative reward and the highest cumulative regret, similar to Scenario 1 in Figure 6 and Figure 10. Thompson sampling with either approximation algorithm requires a larger number of time steps to perform efficient exploration than Scenario 1 in Figure 6 and Figure 10 because of greater number of borrower features. The epsilon-greedy achieves the highest cumulative reward and the lowest cumulative regret; however, the cumulative regret increases linearly as there is a small chance that the algorithm performs exploration in every time step, shown in Figure 14 (d).

2) Scenario 2: Selecting a Single Borrower with Renewal

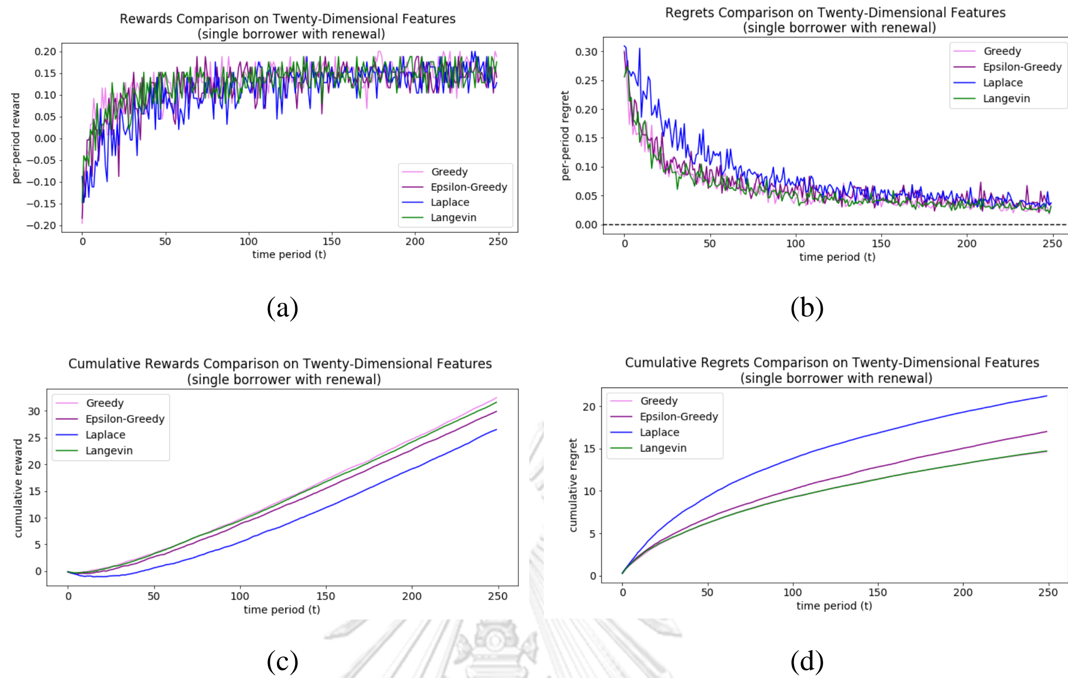


Figure 15. Performance comparisons on twenty-dimensional features (single borrower with renewal)

Figure 15 illustrates the performance of each algorithm when the borrower characteristics are captured by a twenty-dimensional vector, where the setting is that a single borrower is selected per action while the pool of borrowers is renewed in every time step. Due to an increase in the borrower features, Thompson sampling with Laplace approximation needs to perform heavy exploration in a number of time steps before reaching per-period regret comparable to other algorithms. This results in its lowest cumulative reward and highest cumulative regret, shown in Figure 15 (c) and (d). The epsilon-greedy algorithm performs an exploration with a small probability in every time step, resulting in the linear increase in cumulative regret, shown in Figure 15 (d). Because Thompson sampling with Langevin MCMC does not perform such the heavy exploration as Thompson sampling with Laplace approximation in early time steps, evident in Figure 15 (b), the algorithm achieves higher cumulative reward and lower cumulative regret in Figure 15 (c) and (d). Unlike Scenario 2 in Figure 7 and Figure 11, the renewal of borrower pool allows the greedy algorithm to perform better

than Scenario 1; however, because of an increase in borrower features, its performance is not quite different from Thompson sampling with Langevin MCMC.

3) Scenario 3: Selecting Multiple (10) Borrowers without Renewal

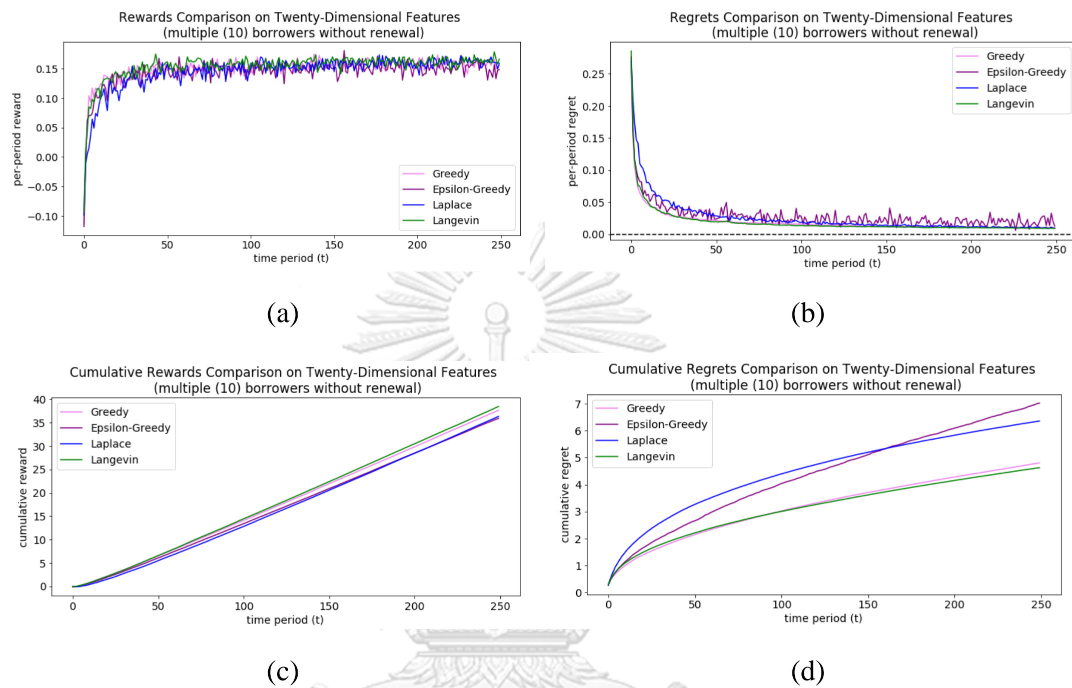


Figure 16. Performance comparisons on twenty-dimensional features (multiple borrowers without renewal)

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

Figure 16 illustrates the performance of each algorithm when the borrower characteristics are captured by a twenty-dimensional vector, where the setting is that multiple (10) borrowers are selected per action while the pool of borrowers is fixed. The epsilon-greedy has a small chance of an exploration in every time step, resulting in the linear increase in the cumulative regret. In Figure 16 (c) and (d), the algorithm results in the lowest cumulative reward and the highest cumulative regret. As the number of borrower features is greater, Thompson sampling with Laplace approximation would perform heavy exploration in a number of time steps before achieving comparable per-period regret in later time steps, shown in Figure 16 (b). As Thompson sampling with Langevin MCMC does not perform quite heavy exploration in early time steps, the algorithm results in higher cumulative reward and lower cumulative regret than Thompson sampling with Laplace approximation, shown in

Figure 16 (c) and (d). As the number of borrower features is greater than Scenario 3 in Figure 8 and Figure 12, Thompson sampling with Langevin MCMC outperforms the greedy algorithm. When the number of borrower features increases, with the same number of borrowers per action, the greedy algorithm would experience difficulties in using such information to find parameters that could generalize well. In this setting, Thompson sampling with Langevin MCMC and the greedy algorithm are not much different in that Langevin MCMC starts at the mode, which is the parameter that the greedy algorithm uses.

4) Scenario 4: Selecting Multiple (10) Borrowers with Renewal

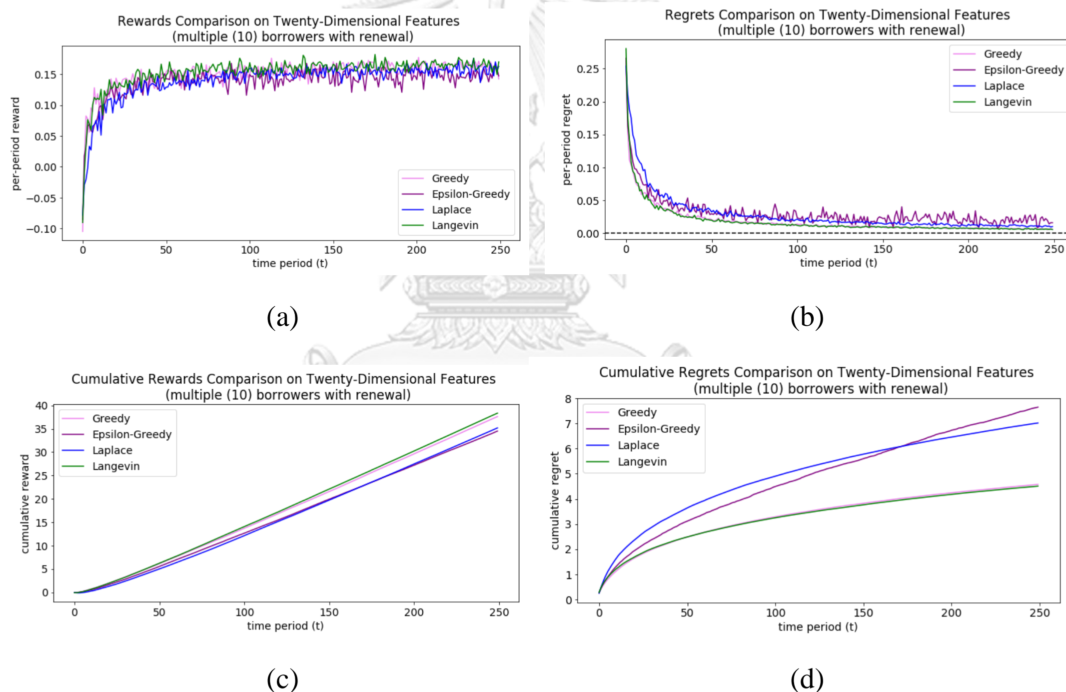


Figure 17. Performance comparisons on twenty-dimensional features (multiple borrowers with renewal)

Figure 17 illustrates the performance of each algorithm when the borrower characteristics are captured by a twenty-dimensional vector, where the setting is that multiple (10) borrowers are selected per action while the pool of borrowers is renewed in every time step. The epsilon-greedy exhibits a linear increase in the cumulative regret, due to a small chance of exploration in every time step, resulting in the lowest cumulative reward and the highest cumulative regret in Figure 17 (c) and (d). As the

number of features capturing borrower characteristics increases, Thompson sampling with Laplace approximation needs to perform heavy exploration in the parameter space before reaching lower level of per-period regret in later time steps, shown in Figure 17 (b). Because Thompson sampling with Langevin MCMC does not perform such the heavy exploration in early time steps, shown in Figure 17 (b), its cumulative reward is larger, and its cumulative regret is lower than Thompson sampling with Laplace approximation. Incorporating two modifications in Scenario 4 would allow the greedy to perform much better than Scenario 1; however, a large number of borrower features cause some difficulties for the greedy algorithm, given the same action specification. In Figure 17 (c) and (d), Thompson sampling with Langevin MCMC performs slightly better than the greedy algorithm. Hence, the efficient exploration by Langevin MCMC in Thompson sampling is still beneficial in the credit scoring setting, when the borrower characteristics are captured by a large number of features.

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

CHAPTER V
CONCLUSION AND DISCUSSION

This study applies the logistic bandit framework, which is a reinforcement learning framework, to credit scoring under the number of feature dimensions $p = 2, 10, 20$. Four different scenarios are studied. Scenario 1 (simple setting) is that the agent would select a single borrower while the pool of borrowers is fixed. Scenario 2 adds a renewal on the group of borrowers in each time step. Scenario 3 allows the agent to choose multiple (10) borrowers. Scenario 4 (credit scoring setting) incorporate both modifications. In this study, reinforcement learning algorithms include the greedy, the epsilon-greedy, Thompson sampling with Laplace approximation and Thompson sampling with Langevin MCMC. To measure the performance of each algorithm under different scenarios, two performance measures, i.e. regrets and rewards, are measured over 250 time steps, averaged on 100 simulation trials. Scenario 1 is a simple logistic bandit environment while Scenario 4 simulates the credit scoring and underwriting processes.

5.1 Conclusion

Cumulative reward is a measure that shows the net financial outcome accumulated from giving out a unit of loans (equally divided among borrowers in case agent gives loan to multiple borrowers) over each time step. Cumulative regret is a measure that shows the true opportunity cost accumulated over each time step. Table 2 and 3 show cumulative rewards and cumulative regrets, respectively, averaged on all simulation trials and computed at the final time step of each algorithm under different feature dimensions, where four scenarios are divided into panels (a) - (d).

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

Table 2. The cumulative rewards of each algorithm under different feature dimensions

Feature Dimensions Algorithm	2-dim	10-dim	20-dim
greedy	33.22	28.78	20.68
epsilon greedy	36.74	32.7	**28.78**
Thompson sampling (Laplace)	**37.74**	**33.44**	27.58
Thompson sampling (Langevin MCMC)	37.47	29.52	22.93

(a) Scenario 1

Feature Dimensions Algorithm	2-dim	10-dim	20-dim
greedy	34.1	**35.9**	**32.44**
epsilon greedy	33.04	32.88	29.86
Thompson sampling (Laplace)	33.36	31.29	26.47
Thompson sampling (Langevin MCMC)	**36.49**	34.48	31.59

(b) Scenario 2

Feature Dimensions Algorithm	2-dim	10-dim	20-dim
greedy	**37.74**	**39.24**	37.64
epsilon greedy	33.4	36.42	35.92
Thompson sampling (Laplace)	35.47	37.71	36.33
Thompson sampling (Langevin MCMC)	36.92	38.96	**38.45**

(c) Scenario 3

Feature Dimensions Algorithm	2-dim	10-dim	20-dim
greedy	**34.44**	**38.58**	37.6
epsilon greedy	30.3	36.3	34.51
Thompson sampling (Laplace)	31.63	36.91	35.19
Thompson sampling (Langevin MCMC)	32.63	38.4	**38.35**

(d) Scenario 4

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

According to Table 2, the cumulative rewards generally decrease as the number of borrower features increases, indicating the increased complexity of the problem that the reinforcement learning agent is facing. In Scenario 1, Thompson sampling with Laplace approximation performs the best in two-dimensional and ten-dimensional cases while the epsilon-greedy performs the best in twenty-dimensional case, followed closely by Thompson sampling with Laplace approximation. This demonstrates the algorithm with efficient exploration performs the best; however, such efficient

exploration incurred higher financial losses than the exploration by epsilon-greedy, especially in beginning time steps. In Scenario 2, Thompson sampling with Langevin MCMC performs the best in two-dimensional case while the greedy algorithm performs the best in other cases. In Scenario 3, the greedy algorithm performs the best in two-dimensional and ten-dimensional cases while Thompson sampling with Langevin MCMC performs the best in twenty-dimensional case. The same conclusion holds in Scenario 4. These results generally illustrate that the greedy algorithm performs better than other algorithms when the setting changes from traditional logistic bandit to credit scoring. Nevertheless, when the number of borrower features increases to twenty, the exploration feature from Langevin MCMC in Thompson sampling leads to slightly better performance than the greedy algorithm.

Table 3. The cumulative regrets of each algorithm under different feature dimensions

Algorithm \ Feature Dimensions	2-dim	10-dim	20-dim
greedy	9.16	18.04	25.11
epsilon greedy	6.28	**13.54**	**17.73**
Thompson sampling (Laplace)	5.13	13.91	19.28
Thompson sampling (Langevin MCMC)	**5.1**	17.83	23.03

(a) Scenario 1

Algorithm \ Feature Dimensions	2-dim	10-dim	20-dim
greedy	**4.81**	**11.13**	**14.63**
epsilon greedy	7.13	12.06	17
Thompson sampling (Laplace)	5.8	14.71	21.21
Thompson sampling (Langevin MCMC)	4.92	11.56	14.71

(b) Scenario 2

Algorithm \ Feature Dimensions	2-dim	10-dim	20-dim
greedy	**0.81**	**3.1**	4.8
epsilon greedy	3.83	5.85	7.01
Thompson sampling (Laplace)	1.34	4.23	6.35
Thompson sampling (Langevin MCMC)	1.2	**3.1**	**4.62**

(c) Scenario 3

จุฬาลงกรณ์มหาวิทยาลัย CHULALONGKORN UNIVERSITY

Algorithm \ Feature Dimensions	2-dim	10-dim	20-dim
greedy	**0.98**	**2.93**	4.57
epsilon greedy	3.43	5.76	7.64
Thompson sampling (Laplace)	1.37	4.39	7.01
Thompson sampling (Langevin MCMC)	1.39	3.06	**4.5**

(d) Scenario 4

Based on Table 3, the cumulative regrets generally increase as the number of borrower features increases. This could be explained by the increased complexity of the problem induced by the greater number of features. In Scenario 1, Thompson sampling with Langevin MCMC performs the best while the epsilon-greedy performs the best in other cases. This result demonstrates that the efficient exploration from Langevin MCMC leads to its best performance; however, with increased number of dimensions, this algorithm requires an exploration in longer time steps, causing the epsilon-greedy algorithm to perform the best. In Scenario 2, the greedy algorithm performs the best in all feature dimensions. In Scenario 3, the greedy algorithm performs the best in two-dimensional and ten-dimensional cases while Thompson sampling with Langevin MCMC performs the best in twenty-dimensional case. Similar results are shown in Scenario 4. Both changes from the traditional logistic bandit to credit scoring provide enough information for the greedy algorithm to select borrowers without the explicit exploration by the algorithm. When the borrower characteristics is captured by twenty dimensional features, the problem agent is facing becomes more complicated, so the exploration from Langevin MCMC in Thompson sampling allows the algorithm to perform slightly better than the greedy algorithm.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

5.2 Discussion

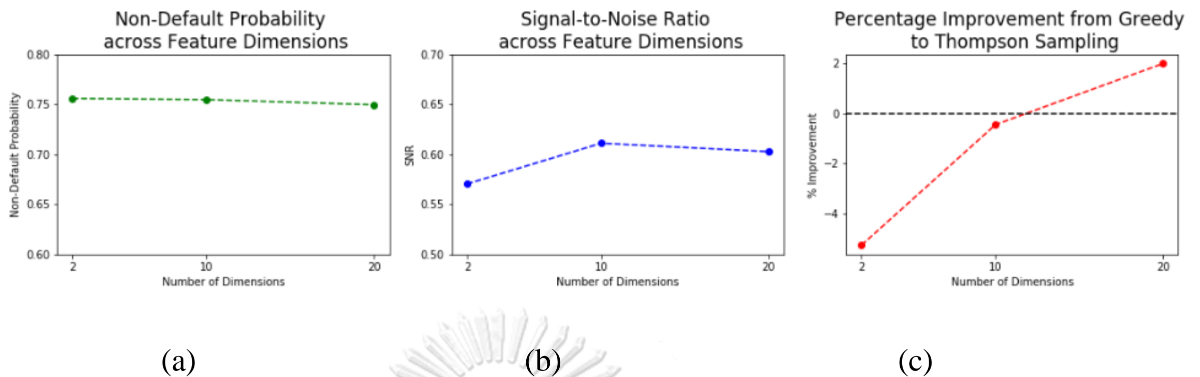
It is commonly believed that an algorithm with an exploration such as the epsilon-greedy would perform better than the greedy (Sutton & Barto, 2018), and an algorithm with an efficient exploration such as Thompson sampling would even perform better than the epsilon-greedy. (Russo et al., 2018) In the simple setting where an agent could grant credit to only one borrower while the pool of borrowers is fixed as in Scenario 1, this statement is true under the small number of dimensions of

borrower features, see Figure 6. An algorithm without explorations would commit too early to an inferior action. An efficient exploration would perform a heavy exploration in the beginning while exploiting the information that agent acquired in later time steps. The algorithm would sacrifice short-term reward in order to increase long-term cumulative rewards. When the number of feature dimensions increases (Figure 10, 14), Thompson sampling algorithms with Laplace approximation and Langevin MCMC need to perform an exploration in many more time steps, resulting in the worse performance in terms of the cumulative reward and the cumulative regret. Still, the greedy algorithm is outperformed by algorithms with an efficient exploration, such as Thompson sampling, supporting the common belief.

Scenario 4 models a more realistic credit scoring and underwriting process using the logistic bandit. The setting includes two modifications: the lender would grant the credit to multiple borrowers per action while the pool of borrowers is renewed in every time step. By incorporating both modifications, in Figure 9, the epsilon-greedy algorithm performs the worst as the cumulative regret increases linearly because of a small chance of the exploration in every time step. Efficient reinforcement learning algorithms, such as Thompson sampling algorithms with Laplace approximation and Langevin MCMC, could achieve better performances because the algorithms perform a heavy exploration in early time steps, but focus on an exploitation in later time steps. However, the greedy algorithm could perform better than Thompson sampling. In contrast to Figure 6, the greedy algorithm does not commit too early to an inferior action under the credit scoring setting in Figure 9. By selecting multiple borrowers per action, the greedy algorithm has enough information to select the parameter without explicitly exploring the action space. When the borrower pool is renewed, a type of exploration is embedded into the learning algorithm without explicitly performing an exploration.

According to Sutton and Barto (2018) and Russo et al. (2018), there is a trade-off between the exploration and the exploitation. The greedy algorithm exploits all the information that the agent currently has in order to select the action with the maximum estimated reward. In contrast, an efficient reinforcement learning algorithm, such as Thompson sampling, would sacrifice per-period reward in early time steps in order to achieve the higher cumulative reward in later time steps. This study has also found that

the exploration is still useful when borrower characteristics are captured by a large number of features.



(a) (b) (c)

Figure 18. Non-default probability, signal-to-noise ratio and percentage improvement across feature dimensions

Figure 18 shows the sample average of non-default probability, the sample average of signal-to-noise ratio and the percentage improvement from the greedy algorithm to Thompson sampling with Langevin MCMC along different feature dimensions under Scenario 4. The percentage improvement uses the cumulative reward at the time step 250 under the setting of credit scoring, i.e. an agent selects multiple (10) borrowers per action while the pool of borrowers is renewed in every time step. Because of the adjustment coefficient, the sample average non-default probabilities and signal-to-noise ratios would be very similar along different dimensions, shown in Figure 18 (a) and (b). When borrower characteristics are described by a two-dimensional vector, based on Figure 9 (c), the percentage improvement is negative, indicating that the credit scoring setting allows the greedy algorithm to perform better than Thompson sampling with Langevin MCMC. However, as the number of dimensions increases to twenty, based on Figure 17 (c), the percentage improvement is positive, implying that an efficient exploration by Langevin MCMC in Thompson sampling is still useful when borrower characteristics are captured by a large number of features, shown in Figure 18 (c). An increase in the borrower features causes difficulties for reinforcement learning algorithms as they require more time steps to perform an exploration. With the same amount of information available for the greedy

algorithm, an increase in borrower features means the greedy would experience difficulties in selecting the parameter which maximizes the cumulative reward.

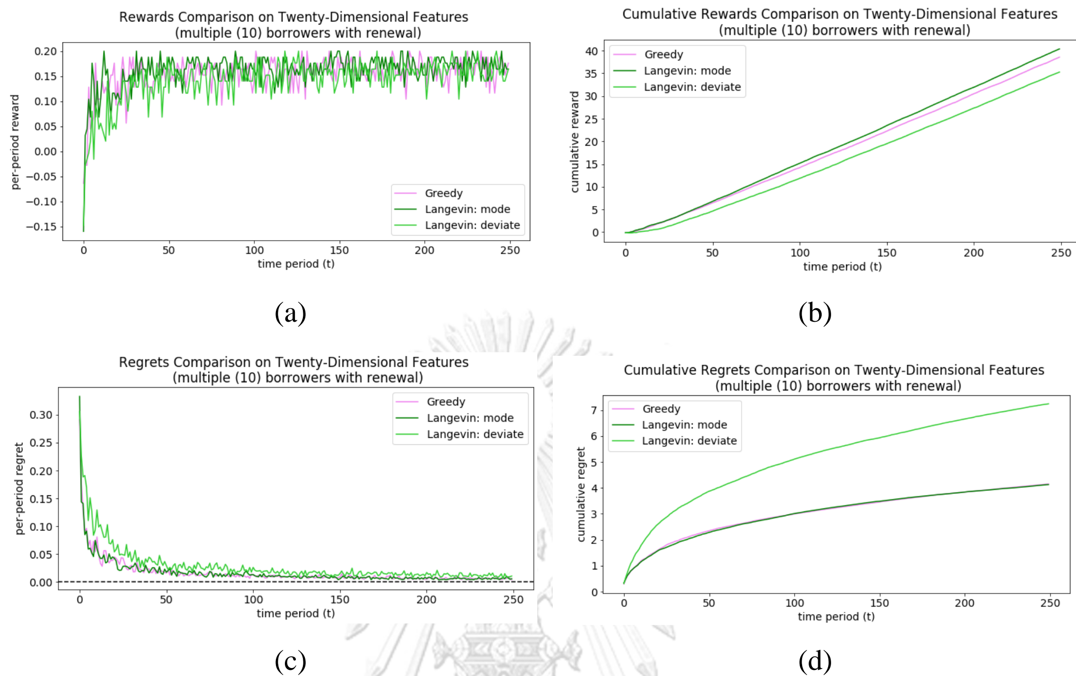


Figure 19. Performance of Thompson sampling with Langevin MCMC when initial parameter is deviated on twenty-dimensional features (multiple borrowers with renewal)

The initialization of Langevin MCMC at posterior mode is crucial for Thompson sampling to work well. Figure 19 shows the performance of such algorithm when the parameter of the model is initialized one standard deviation away from the posterior mode, with respect to each parameter. The setting is that multiple (10) borrowers are selected per action while the pool of borrowers is renewed in every time step, and the borrower characteristics are captured by a twenty-dimensional vector. From Figure 19 (a) and (c), given a particular level of per-period reward or regret, the algorithm with the deviation during initialization needs a larger number of time steps than the algorithm that starts with posterior mode in order to reach that particular level. Therefore, the deviation from posterior mode during the initialization results in lower cumulative reward and higher cumulative regret, shown in Figure 19 (b) and (d).

5.3 Future Research

This study provides a deeper understanding of reinforcement learning towards the logistic bandit under the setting of credit scoring and underwriting. The credit scoring proposed by this study is more complicated than the setting usually studied in reinforcement learning literature. With different settings, the results are different from what is usually found in other reinforcement learning literatures: the greedy approach can outperform Thompson sampling.

Still, this setting is far from the reality of credit scoring. One recommended direction of future research is to evaluate reinforcement learning algorithms in a more complicated setting; for example, when the environment is contextualized, where the optimal action also depends on which group a borrower belongs to. (Russo et al., 2018) When the process of generating ground truth labels given the borrower features is more complicated than the credit scoring model, how would the reinforcement learning algorithms suffer from such complexity, and how would the greedy algorithm perform, compared with reinforcement learning algorithms with efficient exploration?

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

REFERENCES

Dumitrascu, B., Feng, K., & Engelhardt, B. (2018). Pg-ts: Improved thompson sampling for logistic contextual bandits. *Advances in neural information processing systems*, *31*.

Faury, L., Abeille, M., Calauzènes, C., & Fercoq, O. (2020). Improved optimistic algorithms for logistic bandits. International Conference on Machine Learning,

Gamerman, D., & Lopes, H. F. (2006). *Markov chain Monte Carlo: stochastic simulation for Bayesian inference*. CRC press.

Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction* (Vol. 2). Springer.

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications.

Karagulyan, A. (2021). *Sampling with the Langevin Monte-Carlo* Institut Polytechnique de Paris].

Kneiding, C., & Rosenberg, R. (2008). Variations in microcredit interest rates.

Lessmann, S., Baesens, B., Seow, H.-V., & Thomas, L. C. (2015). Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research. *European Journal of Operational Research*, *247*(1), 124-136.

Phillips, R. L. (2018). Pricing credit products. In *Pricing Credit Products*. Stanford University Press.

Rosenberg, R., Gonzalez, A., & Narain, S. (2009). Are microcredit interest rates excessive? *CGAP brief*.

Russo, D. J., Van Roy, B., Kazerouni, A., Osband, I., & Wen, Z. (2018). A tutorial on thompson sampling. *Foundations and Trends® in Machine Learning*, *11*(1), 1-96.

Soch, J., & Allefeld, C. (2018). MACS–a new SPM toolbox for model assessment, comparison and selection. *Journal of neuroscience methods*, *306*, 19-31.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Vojtek, M., & Koèenda, E. (2006). Credit-scoring methods. *Czech Journal of Economics and Finance (Finance a uver)*, *56*(3-4), 152-167.

Wang, J.-K., Lu, C.-J., & Lin, S.-D. (2017). Fast Algorithm for Logistic Bandit.

Zhang, L., Yang, T., Jin, R., Xiao, Y., & Zhou, Z.-H. (2016). Online stochastic linear optimization under one-bit feedback. International Conference on Machine Learning,

CHULALONGKORN UNIVERSITY

Appendix 1: Python code for Bernoulli bandit

A1.1 Package dependencies

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
import scipy
import pandas as pd

seed = 546279
np.random.seed(seed)
```

A1.2 Bernoulli bandit environment

```
class BernoulliBandit:

    def __init__(self, num_groups = 3, true_p = [0.9, 0.8, 0.7]):
        assert len(true_p) == num_groups
        self.num_groups = num_groups
        self.true_p = true_p

    def act(self, action: int):
        assert (action > 0) and (action <= self.num_groups)
        regret = max(self.true_p) - self.true_p[action-1]
        if np.random.random() <= self.true_p[action-1]: # Head
            return 1, regret
        else: # Tail
            return 0, regret
```

A1.3 Greedy and epsilon greedy algorithms

```
class EpsilonGreedy_BernoulliBandit:

    def __init__(self, env: BernoulliBandit, epsilon = 0.05):
        self.env = env
        self.num_groups = self.env.num_groups # same as number of
actions
        self.epsilon = epsilon

    def run(self, horizon = 2000):

        rewards = np.zeros(horizon)
        regrets = np.zeros(horizon)
        # Record # of successes and failures
        num_successes = np.zeros(self.num_groups)
        num_failures = np.zeros(self.num_groups)

        for i in range(horizon):
            if np.random.random() <= self.epsilon: # Choose randomly
                action = np.random.choice(self.num_groups) + 1
            else: # Greedily choosing the optimal one
                p_hat = (num_successes + 1)/(num_successes +
num_failures + 2)
```

```
            max_p = max(p_hat)
            max_idx = np.where(p_hat == max_p)[0]
            action = np.random.choice(max_idx) + 1
        reward, regret = self.env.act(action)

        # Save the values in reward, regret
        rewards[i] += reward
        regrets[i] += regret
        if reward > 0:
            num_successes[action-1] += 1
        else:
            num_failures[action-1] += 1

    # Calculate cumulative regrets and rewards
    cum_rewards = np.cumsum(rewards)
    cum_regrets = np.cumsum(regrets)

    return rewards, cum_rewards, regrets, cum_regrets
```

A1.4 Thompson sampling algorithm and Upper Confidence Bound algorithm

```
def TS_BB(num_successes, num_failures):
    p_hat = np.random.beta(num_successes + 1, num_failures + 1)
    return p_hat

def UCB_BB(num_successes, num_failures):
    t = sum(num_successes + num_failures)
    p_hat = [scipy.stats.beta.ppf(t/(t+1), num_successes[i] + 1,
num_failures[i] + 1) for i in range(len(num_successes))]
    return p_hat

class TS_BernoulliBandit:

    def __init__(self, env: BernoulliBandit, f):
        self.env = env
        self.num_groups = self.env.num_groups
        self.updater = f

    def run(self, horizon = 2000):
        rewards = np.zeros(horizon)
        regrets = np.zeros(horizon)
        # Record # of successes and failures
        num_successes = np.zeros(self.num_groups)
        num_failures = np.zeros(self.num_groups)

        for i in range(horizon):
            p_hat = self.updater(num_successes, num_failures)
            max_p = max(p_hat)
            max_idx = np.where(p_hat == max_p)[0]
            action = np.random.choice(max_idx) + 1
            reward, regret = self.env.act(action)
            # Save the values in reward, regret
            rewards[i] += reward
            regrets[i] += regret
            if reward > 0:
                num_successes[action - 1] += 1
            else:
```

```
                num_failures[action-1] += 1

        # Calculate cumulative regrets and rewards
        cum_rewards = np.cumsum(rewards)
        cum_regrets = np.cumsum(regrets)

        return rewards, cum_rewards, regrets, cum_regrets
```

A1.5 Simulation

```
rewards_g = np.zeros(2000)
cum_rewards_g = np.zeros(2000)
regrets_g = np.zeros(2000)
cum_regrets_g = np.zeros(2000)
n_rounds = 2000

for i in range(n_rounds):
    bb = BernoulliBandit(num_groups = 3, true_p = [0.9, 0.8, 0.7])
    agent = EpsilonGreedy_BernoulliBandit(bb, epsilon = 0)
    a,b,c,d = agent.run()
    rewards_g = rewards_g + a
    cum_rewards_g = cum_rewards_g + b
    regrets_g = regrets_g + c
    cum_regrets_g = cum_regrets_g + d

rewards_g /= n_rounds
cum_rewards_g /= n_rounds
regrets_g /= n_rounds
cum_regrets_g /= n_rounds

rewards = np.zeros(2000)
cum_rewards = np.zeros(2000)
regrets = np.zeros(2000)
cum_regrets = np.zeros(2000)
n_rounds = 2000

for i in range(n_rounds):
    bb = BernoulliBandit(num_groups = 3, true_p = [0.9, 0.8, 0.7])
    agent = EpsilonGreedy_BernoulliBandit(bb, epsilon = 0.05)
    a,b,c,d = agent.run()
    rewards = rewards + a
    cum_rewards = cum_rewards + b
    regrets = regrets + c
    cum_regrets = cum_regrets + d

rewards /= n_rounds
cum_rewards /= n_rounds
regrets /= n_rounds
cum_regrets /= n_rounds

rewards_TS = np.zeros(2000)
cum_rewards_TS = np.zeros(2000)
regrets_TS = np.zeros(2000)
cum_regrets_TS = np.zeros(2000)
n_rounds = 2000

for i in range(n_rounds):
    bb = BernoulliBandit(num_groups = 3, true_p = [0.9, 0.8, 0.7])
```

```
    agent = TS_BernoulliBandit(bb, TS_BB)
    a,b,c,d = agent.run()
    rewards_TS = rewards_TS + a
    cum_rewards_TS = cum_rewards_TS + b
    regrets_TS = regrets_TS + c
    cum_regrets_TS = cum_regrets_TS + d

rewards_TS /= n_rounds
cum_rewards_TS /= n_rounds
regrets_TS /= n_rounds
cum_regrets_TS /= n_rounds

rewards_UCB = np.zeros(2000)
cum_rewards_UCB = np.zeros(2000)
regrets_UCB = np.zeros(2000)
cum_regrets_UCB = np.zeros(2000)
n_rounds = 2000

for i in range(n_rounds):
    bb = BernoulliBandit(num_groups = 3, true_p = [0.9, 0.8, 0.7])
    agent = TS_BernoulliBandit(bb, UCB_BB)
    a,b,c,d = agent.run()
    rewards_UCB = rewards_UCB + a
    cum_rewards_UCB = cum_rewards_UCB + b
    regrets_UCB = regrets_UCB + c
    cum_regrets_UCB = cum_regrets_UCB + d

rewards_UCB /= n_rounds
cum_rewards_UCB /= n_rounds
regrets_UCB /= n_rounds
cum_regrets_UCB /= n_rounds

output_dict = {"greedy_reward": rewards_g, "greedy_cum_reward":
cum_rewards_g, "greedy_regret": regrets_g, "greedy_cum_regret":
cum_regrets_g,
"epsilon_greedy_reward": rewards, "epsilon_greedy_cum_reward":
cum_rewards, "epsilon_greedy_regret": regrets,
"epsilon_greedy_cum_regret": cum_regrets,
              "TS_reward": rewards_TS, "TS_cum_reward":
cum_rewards_TS, "TS_regret": regrets_TS, "TS_cum_regret":
cum_regrets_TS,
              "UCB_reward": rewards_UCB, "UCB_cum_reward":
cum_rewards_UCB, "UCB_regret": regrets_UCB, "UCB_cum_regret":
cum_regrets_UCB}
df = pd.DataFrame(output_dict)
df.to_csv("Bernoulli_Bandit_2000.csv", index = False)
```

A1.6 Visualizations

```
df = pd.read_csv("Bernoulli_Bandit_2000.csv")
rewards_g = df["greedy_reward"]
rewards = df["epsilon_greedy_reward"]
rewards_TS = df["TS_reward"]
rewards_UCB = df["UCB_reward"]
cum_rewards_g = df["greedy_cum_reward"]
cum_rewards = df["epsilon_greedy_cum_reward"]
cum_rewards_TS = df["TS_cum_reward"]
```

```
cum_rewards_UCB = df["UCB_cum_reward"]
regrets_g = df["greedy_regret"]
regrets = df["epsilon_greedy_regret"]
regrets_TS = df["TS_regret"]
regrets_UCB = df["UCB_regret"]
cum_regrets_g = df["greedy_cum_regret"]
cum_regrets = df["epsilon_greedy_cum_regret"]
cum_regrets_TS = df["TS_cum_regret"]
cum_regrets_UCB = df["UCB_cum_regret"]

color_scheme = {"Greedy": "purple", "TS": "orange", "Greedy0":
"violet", "UCB": "deepskyblue"}

plt.rcParams.update({'font.size': 14})
fig, ax = plt.subplots(4,1, figsize = (10, 20))
ax[0].plot(np.arange(len(rewards)), rewards_g, label = "Greedy",
color = color_scheme["Greedy0"])
ax[0].plot(np.arange(len(rewards)), rewards, label = "Epsilon-
Greedy", color = color_scheme["Greedy"])
ax[0].plot(np.arange(len(rewards)), rewards_TS, label = "TS", color =
color_scheme["TS"])
ax[0].plot(np.arange(len(rewards)), rewards_UCB, label = "UCB", color
= color_scheme["UCB"])
ax[0].legend()
ax[0].set(title = f"Rewards Comparison: Bernoulli Bandit", xlabel =
"time period (t)", ylabel = "per-period reward")
ax[1].plot(np.arange(len(rewards)), cum_rewards_g, label = "Greedy",
color = color_scheme["Greedy0"])
ax[1].plot(np.arange(len(rewards)), cum_rewards, label = "Epsilon-
Greedy", color = color_scheme["Greedy"])
ax[1].plot(np.arange(len(rewards)), cum_rewards_TS, label = "TS",
color = color_scheme["TS"])
ax[1].plot(np.arange(len(rewards)), cum_rewards_UCB, label = "UCB",
color = color_scheme["UCB"])
ax[1].set(title = f"Cumulative Rewards Comparison: Bernoulli Bandit",
xlabel = "time period (t)", ylabel = "cumulative reward")
ax[1].legend()
ax[2].plot(np.arange(len(rewards)), regrets_g, label = "Greedy",
color = color_scheme["Greedy0"])
ax[2].plot(np.arange(len(rewards)), regrets, label = "Epsilon-
Greedy", color = color_scheme["Greedy"])
ax[2].plot(np.arange(len(rewards)), regrets_TS, label = "TS", color =
color_scheme["TS"])
ax[2].plot(np.arange(len(rewards)), regrets_UCB, label = "UCB", color
= color_scheme["UCB"])
ax[2].axhline(y = 0, color = 'black', linestyle = '--')
ax[2].set(title = f"Regrets Comparison: Bernoulli Bandit", xlabel =
"time period (t)", ylabel = "per-period regret")
ax[2].legend()
ax[3].plot(np.arange(len(rewards)), cum_regrets_g, label = "Greedy",
color = color_scheme["Greedy0"])
ax[3].plot(np.arange(len(rewards)), cum_regrets, label = "Epsilon-
Greedy", color = color_scheme["Greedy"])
ax[3].plot(np.arange(len(rewards)), cum_regrets_TS, label = "TS",
color = color_scheme["TS"])
ax[3].plot(np.arange(len(rewards)), cum_regrets_UCB, label = "UCB",
color = color_scheme["UCB"])
```

```
ax[3].set(title = f"Cumulative Regrets Comparison: Bernoulli Bandit",
xlabel = "time period (t)", ylabel = "cumulative regret")
ax[3].legend()
plt.tight_layout()
plt.show()
```

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

Appendix 2: Python code for logistic bandit

A2.1 Dependencies

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
import scipy
import pandas as pd

seed = 491372
n_rounds = 1

np.random.seed(seed)

top_str = "10"
refresh_str = "refresh"

top_display = "multiple (10) borrowers"
refresh_display = "with renewal"
```

A2.2 Logistic bandit environment

Parameters:

- num_borrowers (default: 100) is the number of borrowers applying for loan in each iteration

- dim (default: 2) is the number of feature dimensions (excluding intercept)

- sigma_p (default: 1) is the standard deviation of parameter beta, representing the uncertainty of environment

- gain (default: 0.2) is the interest rate that borrower pays when he does not default

- loss (default: -1) is the loss incurred to the lender when borrower defaults

- intercept (default: 1.5) is the actual beta0, implying the average non-default probability.

- refresh (default: False) represents whether the group of borrowers is renewed after each iteration.

- top_k (default: 1) indicates the number of borrowers who would get loan in each iteration.

Methods:

- get_borrowers_attributes(): Get the features from the environment

- act(action): given the weight given to each borrower (total sum = 1), return the observation, rewards and regrets

```
class LogisticBandit:
    def __init__(self, num_borrowers = 100, dim = 2, sigma_p = 1,
gain = 0.2, loss = -1, intercept = 1.5, refresh = False, top_k = 1):
        # Record all parameters as instance variables
        self.num_borrowers = num_borrowers
        self.dim = dim
        self.sigma_p = sigma_p
        self.gain = gain
        self.loss = loss
        self.intercept = intercept
        self.refresh = refresh
        self.top_k = top_k
        # Generate ground-truth beta based on the given dimension and
given SD (sigma_p)
        self.beta = sigma_p * np.random.randn(dim, 1)
        # Generate borrower features based on the number of
dimensions and number of borrowers
        self.feature = np.random.randn(num_borrowers, dim) *
np.sqrt(2/self.dim)
        # If intercept is not zero, increment the dimension,
concatenate intercept with beta and concatenate ones with features
        if intercept != 0:
            self.dim += 1
            self.beta = np.concatenate(([[intercept]], self.beta))
            self.feature =
np.concatenate((np.ones((num_borrowers,1)), self.feature), axis = 1)
        # Calculate the true probabilities of non-default recorded as
true_ps
        exp_logits = np.exp(np.matmul(self.feature, self.beta))
        self.true_ps = exp_logits/(1 + exp_logits)
        # Calculate the expected rewards based on true_ps and gain
with loss
        exp_rewards = np.dot(self.true_ps, self.gain) + np.dot(1-
self.true_ps, self.loss)
        # Calculate top k average reward as the maximum expected
reward over all possible actions
        top_k_avg = np.sum(-np.sort(-
exp_rewards[:,0])[:self.top_k])/self.top_k
        # Calculate regret of each borrower as top_k_avg subtracted
by expected reward of each borrower
        self.regrets = top_k_avg - exp_rewards

    def get_borrowers_attributes(self):
        return self.feature

    def act(self, action):
        # Assert the equality of dimension and the sum of all weights
must be equal to one
        assert (len(action) == self.num_borrowers) and
(abs(sum(action)-1) < 0.0005)
        # Randomize the default and non-default outcome based on
true_ps
        # 1 represents non-default while 0 represents default
        actual_observations =
np.where(np.random.random((self.num_borrowers, 1)) <= self.true_ps,
1, 0)
        # Assign the returns of each borrower based on
actual_observations to the gain and loss
```

```
        returns = np.where(actual_observations == 1, self.gain,
self.loss)
        # Get the actual reward by dot product the returns and the
actions (representing the weight)
        actual_rewards = np.dot(np.squeeze(returns), action)
        # Get the actual regret by dot product regrets with the
action
        actual_regrets = np.dot(np.squeeze(self.regrets), action)
        # If refresh = TRUE, then randomize the feature, calculate
true_ps, expected rewards, regrets again
        if self.refresh:
            if self.intercept == 0:
                self.feature = np.random.randn(self.num_borrowers,
self.dim)  * np.sqrt(2/self.dim)
            else:
                self.feature = np.random.randn(self.num_borrowers,
self.dim-1)  * np.sqrt(2/self.dim)
                self.feature =
np.concatenate((np.ones((self.num_borrowers,1)), self.feature), axis
= 1)
            exp_logits = np.exp(np.matmul(self.feature, self.beta))
            self.true_ps = exp_logits/(1 + exp_logits)
            exp_rewards = np.dot(self.true_ps, self.gain) + np.dot(1-
self.true_ps, self.loss)
            top_k_avg = np.sum(-np.sort(-
exp_rewards[:,0])[:self.top_k])/self.top_k
            self.regrets = top_k_avg - exp_rewards
        # Return the actual observations, actual rewards and actual
regrets
        return np.squeeze(actual_observations), actual_rewards,
actual_regrets
```

A2.3 Greedy and epsilon-greedy algorithms

Parameters:

- env: LogisticBandit

- epsilon (default: 0.05) : the probability that the agent would explore on any one of all possible actions

Methods:

- run(horizon = 250): Perform the algorithm in the number of time steps (horizon) and return rewards, cumulative rewards, regrets and cumulative regrets in each time step

```
# Helper Function: Find the index of top k from an array x
# Source: https://stackoverflow.com/questions/6910641/how-do-i-get-
indices-of-n-maximum-values-in-a-numpy-array

def find_top_k_idx(x, k):
    top_k_idx = np.argpartition(x, -k)[-k:]
    return top_k_idx
```

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

```
class EpsilonGreedy_LogisticBandit:

    def __init__(self, env: LogisticBandit, epsilon = 0.05):
        # Record all parameters as instance variables
        self.env = env
        self.epsilon = epsilon
        # Record necessary variables: dimension, number of borrowers,
feature top_k from the environment
        self.dim = self.env.dim
        self.num_borrowers  = self.env.num_borrowers
        self.feature = self.env.get_borrowers_attributes()
        self.top_k = self.env.top_k
        # The solver is LogisticRegression with L2 regularization and
C depends on sigma_p from the environment
        self.solver = LogisticRegression(penalty='l2',
                                         C=2*(self.env.sigma_p**2),
                                         fit_intercept=False,
                                         warm_start=True)

    def run(self, horizon = 250):
        # Set initialization phase to True and it will be false if
there exists at least one observation of both classes
        initialization_phase = True
        # Initialize rewards and regrets
        rewards = np.zeros(horizon)
        regrets = np.zeros(horizon)
        # Initialize theta
        theta = np.zeros(self.dim)
        # Record the previous observations on X and y
        X = np.zeros((horizon*self.top_k, self.dim))
        y = np.zeros(horizon*self.top_k)

        for i in range(horizon):
            # If in initialization phrase or exploration, then choose
one action randomly
            if initialization_phase or np.random.random() <=
self.epsilon:
                action_idx = np.random.choice(self.num_borrowers,
size = self.top_k, replace = False)
            # Conduct an exploitation
            else:
                # Fit the Logistic Regression on all previous
observations
                self.solver.fit(X[:self.top_k*(i+1),],
y[:self.top_k*(i+1)])
                # Get theta from the solver
                theta = self.solver.coef_.reshape(self.dim)
                # Find the index of top_k from the given feature and
estimated theta
                action_idx = find_top_k_idx(np.dot(self.feature,
theta), self.top_k)
            # Convert the index of targeted borrowers to the weight
            action = [(1/self.top_k) if j in action_idx else 0 for j
in range(self.num_borrowers)]
            # Get observation, reward and regret from interacting
with the environment
            observation, reward, regret = self.env.act(action)
```

```
            # Get start_idx and end_idx, and assign the additional
features and observation
            start_idx = i * self.top_k
            end_idx = (i+1) * self.top_k
            X[start_idx:end_idx,:] = self.feature[action_idx, :]
            y[start_idx:end_idx] = observation[action_idx]
            # Assign the reward and regret at the i-th timestep
            rewards[i] = reward
            regrets[i] = regret
            # Get the new feature if refresh is TRUE
            if self.env.refresh:
                self.feature = self.env.get_borrowers_attributes()
            # If we have both 0 and 1 class, stop the initialization
phase
            if initialization_phase and len(np.unique(y[:end_idx]))
== 2:
                initialization_phase = False

        # Calculate cumulative regrets and rewards
        cum_rewards = np.cumsum(rewards)
        cum_regrets = np.cumsum(regrets)

        return rewards, cum_rewards, regrets, cum_regrets
```

A2.4 Thompson sampling with Laplace / Langevin MCMC

Parameters:

- env: LogisticBandit
- f : the updater function (either LaplaceTS_LB or LangevinTS_LB)

Methods:

- run(horizon = 200): Perform the algorithm in the number of time steps (horizon) and return rewards, cumulative rewards, regrets and cumulative regrets in each time step

Chulalongkorn University

```
# Helper function: calculate Logistic Log Likelihood given normal
prior
def Logistic_Log_Likelihood(X, y, sigma_p_squared, beta, prior_mean):
    prior_constant = -
0.5*beta.shape[0]*np.log(2*np.pi*sigma_p_squared)
    prior_ll = -np.dot((beta-prior_mean).T, (beta-
prior_mean))/(2*sigma_p_squared)
    logistic_ll = np.sum(y*np.dot(X, beta) -
np.log(1+np.exp(np.dot(X, beta))))
    return prior_constant + prior_ll + logistic_ll

# Helper function: calculate the first derivative of Logistic
Likelihood given normal prior
def Logistic_Log_Likelihood_D1(X, y, sigma_p_squared, beta,
prior_mean):
    prior_D1 = -(beta-prior_mean)/(sigma_p_squared)
    y_hat = 1/(1+np.exp(-np.dot(X, beta)))
```

```
    logistic_D1 = np.dot(X.T, (y - y_hat))
    return prior_D1 + logistic_D1

# Helper function: calculate the second derivative of Logistic
Likelihood given normal prior
def Logistic_Log_Likelihood_D2(X, y, sigma_p_squared, beta,
prior_mean):
    prior_D2 = -np.eye(beta.shape[0])/sigma_p_squared
    y_hat = 1/(1+np.exp(-np.dot(X, beta)))
    D = np.diag(np.multiply(y_hat, 1-y_hat).reshape(y_hat.shape[0]))
    logistic_D2 = -np.dot(np.dot(X.T, D), X)
    return prior_D2 + logistic_D2


# Helper function: Sample beta_hat using Laplace approximation
def LaplaceTS_LB(X, y, sigma_p_squared, beta, prior_mean):
    mode = beta.reshape(beta.shape[0])
    cov = -np.linalg.inv(Logistic_Log_Likelihood_D2(X, y,
sigma_p_squared, beta, prior_mean))
    beta_hat = np.random.multivariate_normal(mode, cov)
    return beta_hat


# Helper function: Sample beta_hat using Langevin MCMC
def LangevinTS_LB(X, y, sigma_p_squared, beta, prior_mean, step_size
= 2, num_steps = 100):

    beta_hat = beta
    A_inv = -Logistic_Log_Likelihood_D2(X, y, sigma_p_squared, beta,
prior_mean)
    A = np.linalg.inv(A_inv)
    A_sqrt = scipy.linalg.sqrtm(A)

    for k in range(num_steps):
        epsilon = np.random.normal(size = beta.shape)
        proposal = beta_hat + step_size*(np.dot(A,
Logistic_Log_Likelihood_D1(X, y, sigma_p_squared, beta_hat,
prior_mean))) + np.sqrt(2*step_size)*np.dot(A_sqrt, epsilon)
        upper_tmp = beta_hat - proposal - step_size*np.dot(A,
Logistic_Log_Likelihood_D1(X, y, sigma_p_squared, proposal,
prior_mean))
        upper = Logistic_Log_Likelihood(X, y, sigma_p_squared,
proposal, prior_mean)-(1/(4*step_size))*np.dot(upper_tmp.T,
np.dot(A_inv, upper_tmp))
        lower_tmp = proposal - beta_hat - step_size*np.dot(A,
Logistic_Log_Likelihood_D1(X, y, sigma_p_squared, beta_hat,
prior_mean))
        lower = Logistic_Log_Likelihood(X, y, sigma_p_squared,
beta_hat, prior_mean)-(1/(4*step_size))*np.dot(lower_tmp.T,
np.dot(A_inv, lower_tmp))

        p_accept = np.minimum(1, np.exp(upper-lower))

        if np.random.random() <= p_accept:
            beta_hat = proposal

    return beta_hat
```

```
class TS_LogisticBandit:

    def __init__(self, env: LogisticBandit, f):
        # Record environment with its associated variables and
updater function
        self.env = env
        self.dim = self.env.dim
        self.num_borrowers  = self.env.num_borrowers
        self.feature = self.env.get_borrowers_attributes()
        self.top_k = self.env.top_k
        # The solver is Logistic Regression with L2 penalty
        self.solver = LogisticRegression(penalty='l2',
                                         C=2*(self.env.sigma_p**2),
                                         fit_intercept=False,
                                         warm_start=True)
        self.updater = f

    def run(self, horizon = 200):
        # Set initialization phase to True
        initialization_phase = True
        # Initialize rewards and regrets
        rewards = np.zeros(horizon)
        regrets = np.zeros(horizon)
        # Initialize theta and prior mean
        theta = np.zeros((self.dim,1))
        prior_mean = np.zeros((self.dim, 1))
        # Record all previous observations in X and y
        X = np.zeros((horizon*self.top_k, self.dim))
        y = np.zeros((horizon*self.top_k,1))

        for i in range(horizon):
            # If in the initialization phase, choose one of all
possible actions randomly
            if initialization_phase:
                action_idx = np.random.choice(self.num_borrowers,
size = self.top_k, replace = False)
            # If not in the initialization phase, find the action idx
that exploits the information
            else:
                # Fit the solver with all previous information
                self.solver.fit(X[:self.top_k*(i+1),],
y[:self.top_k*(i+1),].reshape(self.top_k*(i+1)))
                # Get the solver coefficient as mode
                mode = self.solver.coef_.T
                # Get the theta from either Laplace Approximation or
Langevin MCMC (in updater)
                theta = self.updater(X[:self.top_k*(i+1),],
y[:self.top_k*(i+1),], self.env.sigma_p**2, mode, prior_mean)
                # Choose index of k borrowers by using the observed
features and estimated theta
                action_idx = find_top_k_idx(np.dot(self.feature,
theta.reshape(self.dim)), self.top_k)
            # Convert the index of k borrowers into the action as
weight
            action = [(1/self.top_k) if j in action_idx else 0 for j
in range(self.num_borrowers)]
            # Get observation, reward and regret from the interaction
with environment
```

```
            observation, reward, regret = self.env.act(action)
            # Update the observation on X and y
            start_idx = i * self.top_k
            end_idx = (i+1) * self.top_k
            X[start_idx:end_idx,] = self.feature[action_idx,]
            y[start_idx:end_idx,0] = observation[action_idx]
            # Record the reward and regret on the i-th timestep
            rewards[i] = reward
            regrets[i] = regret
            # If refresh is TRUE, get the feature again from the
environment
            if self.env.refresh:
                self.feature = self.env.get_borrowers_attributes()
            # If we have both 0 and 1 stop the initialization phase
            if initialization_phase and len(np.unique(y[:end_idx,]))
== 2:
                initialization_phase = False

        # Calculate cumulative regrets and rewards
        cum_rewards = np.cumsum(rewards)
        cum_regrets = np.cumsum(regrets)

        return rewards, cum_rewards, regrets, cum_regrets
```

A2.5 Simulation

```
num_rounds = 100

rewards_g = np.zeros((250, num_rounds))
cum_rewards_g = np.zeros((250, num_rounds))
regrets_g = np.zeros((250, num_rounds))
cum_regrets_g = np.zeros((250, num_rounds))

for i in range(num_rounds):
    if i%10 == 0:
        print(i)
    lb = LogisticBandit(refresh = True, top_k = 10, dim = 20)
    agent = EpsilonGreedy_LogisticBandit(lb, epsilon = 0)
    a,b,c,d = agent.run(horizon = 250)
    rewards_g[:,i] = a
    cum_rewards_g[:,i] = b
    regrets_g[:,i] = c
    cum_regrets_g[:,i] = d

x = pd.DataFrame(rewards_g)
x.to_csv(f"greedy_reward_top_{top_str}_{refresh_str}_20D_R{n_rounds}.
csv", index = False)
x = pd.DataFrame(cum_rewards_g)
x.to_csv(f"greedy_cum_reward_top_{top_str}_{refresh_str}_20D_R{n_roun
ds}.csv", index = False)
x = pd.DataFrame(regrets_g)
x.to_csv(f"greedy_regret_top_{top_str}_{refresh_str}_20D_R{n_rounds}.
csv", index = False)
x = pd.DataFrame(cum_regrets_g)
x.to_csv(f"greedy_cum_regret_top_{top_str}_{refresh_str}_20D_R{n_roun
ds}.csv", index = False)
```

```
rewards = np.zeros((250, num_rounds))
cum_rewards = np.zeros((250, num_rounds))
regrets = np.zeros((250, num_rounds))
cum_regrets = np.zeros((250, num_rounds))

for i in range(num_rounds):
    if i%10 == 0:
        print(i)
    lb = LogisticBandit(refresh = True, top_k = 10, dim = 20)
    agent = EpsilonGreedy_LogisticBandit(lb, epsilon = 0.05)
    a,b,c,d = agent.run(horizon = 250)
    rewards[:,i] = a
    cum_rewards[:,i] = b
    regrets[:,i] = c
    cum_regrets[:,i] = d

x = pd.DataFrame(rewards)
x.to_csv(f"epsilon_greedy_reward_top_{top_str}_{refresh_str}_20D_R{n_
rounds}.csv", index = False)
x = pd.DataFrame(cum_rewards)
x.to_csv(f"epsilon_greedy_cum_reward_top_{top_str}_{refresh_str}_20D_
R{n_rounds}.csv", index = False)
x = pd.DataFrame(regrets)
x.to_csv(f"epsilon_greedy_regret_top_{top_str}_{refresh_str}_20D_R{n_
rounds}.csv", index = False)
x = pd.DataFrame(cum_regrets)
x.to_csv(f"epsilon_greedy_cum_regret_top_{top_str}_{refresh_str}_20D_
R{n_rounds}.csv", index = False)

rewards_LTS = np.zeros((250, num_rounds))
cum_rewards_LTS = np.zeros((250, num_rounds))
regrets_LTS = np.zeros((250, num_rounds))
cum_regrets_LTS = np.zeros((250, num_rounds))

for i in range(num_rounds):
    if i%10 == 0:
        print(i)
    lb = LogisticBandit(refresh = True, top_k = 10, dim = 20)
    agent = TS_LogisticBandit(lb, LaplaceTS_LB)
    a,b,c,d = agent.run(horizon = 250)
    rewards_LTS[:,i] = a
    cum_rewards_LTS[:,i] = b
    regrets_LTS[:,i] = c
    cum_regrets_LTS[:,i] = d

x = pd.DataFrame(rewards_LTS)
x.to_csv(f"thompson_laplace_reward_top_{top_str}_{refresh_str}_20D_R{
n_rounds}.csv", index = False)
x = pd.DataFrame(cum_rewards_LTS)
x.to_csv(f"thompson_laplace_cum_reward_top_{top_str}_{refresh_str}_20
D_R{n_rounds}.csv", index = False)
x = pd.DataFrame(regrets_LTS)
x.to_csv(f"thompson_laplace_regret_top_{top_str}_{refresh_str}_20D_R{
n_rounds}.csv", index = False)
x = pd.DataFrame(cum_regrets_LTS)
x.to_csv(f"thompson_laplace_cum_regret_top_{top_str}_{refresh_str}_20
D_R{n_rounds}.csv", index = False)
```

```
rewards_L = np.zeros((250, num_rounds))
cum_rewards_L = np.zeros((250, num_rounds))
regrets_L = np.zeros((250, num_rounds))
cum_regrets_L = np.zeros((250, num_rounds))

for i in range(num_rounds):
    if i%10 == 0:
        print(i)
    lb = LogisticBandit(refresh = True, top_k = 10, dim = 20)
    agent = TS_LogisticBandit(lb, LangevinTS_LB)
    a,b,c,d = agent.run(horizon = 250)
    rewards_L[:,i] = a
    cum_rewards_L[:,i] = b
    regrets_L[:,i] = c
    cum_regrets_L[:,i] = d

x = pd.DataFrame(rewards_L)
x.to_csv(f"thompson_langevin_reward_top_{top_str}_{refresh_str}_20D_R
{n_rounds}.csv", index = False)
x = pd.DataFrame(cum_rewards_L)
x.to_csv(f"thompson_langevin_cum_reward_top_{top_str}_{refresh_str}_2
0D_R{n_rounds}.csv", index = False)
x = pd.DataFrame(regrets_L)
x.to_csv(f"thompson_langevin_regret_top_{top_str}_{refresh_str}_20D_R
{n_rounds}.csv", index = False)
x = pd.DataFrame(cum_regrets_L)
x.to_csv(f"thompson_langevin_cum_regret_top_{top_str}_{refresh_str}_2
0D_R{n_rounds}.csv", index = False)
```

A2.6 Visualizations

```
def get_avg_output(df_initial):
    filename = f"{df_initial}_R1.csv"
    df = pd.read_csv(filename)
    n = df.shape[1]
    rowsum = df.sum(axis = "columns")
    return rowsum/n

rewards_g =
get_avg_output(f"greedy_reward_top_{top_str}_{refresh_str}_20D")
cum_rewards_g =
get_avg_output(f"greedy_cum_reward_top_{top_str}_{refresh_str}_20D")
regrets_g =
get_avg_output(f"greedy_regret_top_{top_str}_{refresh_str}_20D")
cum_regrets_g =
get_avg_output(f"greedy_cum_regret_top_{top_str}_{refresh_str}_20D")

rewards =
get_avg_output(f"epsilon_greedy_reward_top_{top_str}_{refresh_str}_20
D")
cum_rewards =
get_avg_output(f"epsilon_greedy_cum_reward_top_{top_str}_{refresh_str
}_20D")
regrets =
get_avg_output(f"epsilon_greedy_regret_top_{top_str}_{refresh_str}_20
D")
```

```
cum_regrets =
get_avg_output(f"epsilon_greedy_cum_regret_top_{top_str}_{refresh_str
}_20D")

rewards_LTS =
get_avg_output(f"thompson_laplace_reward_top_{top_str}_{refresh_str}_
20D")
cum_rewards_LTS =
get_avg_output(f"thompson_laplace_cum_reward_top_{top_str}_{refresh_s
tr}_20D")
regrets_LTS =
get_avg_output(f"thompson_laplace_regret_top_{top_str}_{refresh_str}_
20D")
cum_regrets_LTS =
get_avg_output(f"thompson_laplace_cum_regret_top_{top_str}_{refresh_s
tr}_20D")

rewards_L =
get_avg_output(f"thompson_langevin_reward_top_{top_str}_{refresh_str}
_20D")
cum_rewards_L =
get_avg_output(f"thompson_langevin_cum_reward_top_{top_str}_{refresh_
str}_20D")
regrets_L =
get_avg_output(f"thompson_langevin_regret_top_{top_str}_{refresh_str}
_20D")
cum_regrets_L =
get_avg_output(f"thompson_langevin_cum_regret_top_{top_str}_{refresh_
str}_20D")

color_scheme = {"Greedy": "purple", "Laplace": "blue", "Langevin":
"green", "Greedy0": "violet"}
plt.rcParams.update({'font.size': 14})
fig, ax = plt.subplots(4,1, figsize = (10, 20))
ax[0].plot(np.arange(len(rewards)), rewards_g, label = "Greedy",
color = color_scheme["Greedy0"])
ax[0].plot(np.arange(len(rewards)), rewards, label = "Epsilon-
Greedy", color = color_scheme["Greedy"])
ax[0].plot(np.arange(len(rewards)), rewards_LTS, label = "Laplace",
color = color_scheme["Laplace"])
ax[0].plot(np.arange(len(rewards)), rewards_L, label = "Langevin",
color = color_scheme["Langevin"])
ax[0].legend()
ax[0].set(title = f"Rewards Comparison on Twenty-Dimensional Features
\n ({top_display} {refresh_display})", xlabel = "time period (t)",
ylabel = "per-period reward")
ax[1].plot(np.arange(len(rewards)), cum_rewards_g, label = "Greedy",
color = color_scheme["Greedy0"])
ax[1].plot(np.arange(len(rewards)), cum_rewards, label = "Epsilon-
Greedy", color = color_scheme["Greedy"])
ax[1].plot(np.arange(len(rewards)), cum_rewards_LTS, label =
"Laplace", color = color_scheme["Laplace"])
ax[1].plot(np.arange(len(rewards)), cum_rewards_L, label =
"Langevin", color = color_scheme["Langevin"])
ax[1].set(title = f"Cumulative Rewards Comparison on Twenty-
Dimensional Features \n ({top_display} {refresh_display})", xlabel =
"time period (t)", ylabel = "cumulative reward")
ax[1].legend()
```

```
ax[2].plot(np.arange(len(rewards)), regrets_g, label = "Greedy",
color = color_scheme["Greedy0"])
ax[2].plot(np.arange(len(rewards)), regrets, label = "Epsilon-
Greedy", color = color_scheme["Greedy"])
ax[2].plot(np.arange(len(rewards)), regrets_LTS, label = "Laplace",
color = color_scheme["Laplace"])
ax[2].plot(np.arange(len(rewards)), regrets_L, label = "Langevin",
color = color_scheme["Langevin"])
ax[2].axhline(y = 0, color = 'black', linestyle = '--')
ax[2].set(title = f"Regrets Comparison on Twenty-Dimensional Features
\n ({top_display} {refresh_display})", xlabel = "time period (t)",
ylabel = "per-period regret")
ax[2].legend()
ax[3].plot(np.arange(len(rewards)), cum_regrets_g, label = "Greedy",
color = color_scheme["Greedy0"])
ax[3].plot(np.arange(len(rewards)), cum_regrets, label = "Epsilon-
Greedy", color = color_scheme["Greedy"])
ax[3].plot(np.arange(len(rewards)), cum_regrets_LTS, label =
"Laplace", color = color_scheme["Laplace"])
ax[3].plot(np.arange(len(rewards)), cum_regrets_L, label =
"Langevin", color = color_scheme["Langevin"])
ax[3].set(title = f"Cumulative Regrets Comparison on Twenty-
Dimensional Features \n ({top_display} {refresh_display})", xlabel =
"time period (t)", ylabel = "cumulative regret")
ax[3].legend()
plt.tight_layout()
plt.show()
```

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

VITA

NAME	Kantapong Visantavarakul
DATE OF BIRTH	1 Jan 1999
PLACE OF BIRTH	Bangkok, Thailand
INSTITUTIONS ATTENDED	Bachelor of Economics, Thammasat University (First Class Honours)

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University