Student Specific Smart Question Recommender

Shashank P
Dept. of Computer Science and Engineering
National Institute of Technology Karnataka,
Surathkal, Mangalore, India
shashankp5424@gmail.com

Praveen Kumar Gupta
Dept. of Computer Science and Engineering
National Institute of Technology Karnataka, Surathkal,
Mangalore, India
pvgupta24@gmail.com

K. Chandrasekaran
Dept. of Computer Science and Engineering
National Institute of Technology Karnataka, Surathkal,
Mangalore, India
kch@nitk.ac.in

***Abstract* - A common difficulty that a student faces when it comes to studying is personalised attention and gradual improvement in the learning process. To test one's knowledge and thoroughness in a subject, students take tests and mock exams. These tests and mock exams help the student to analyse and realise his / her strengths and weaknesses. But these tests and mock exams do very less in directly trying to improve the student's knowledge in the subject. Sometimes these mock tests can be either very easy or too difficult.**

To solve this problem of biased (too difficult or very easy) set of questions in tests, mock exams or any practice paper, a naive approach is to look at the performance of students after the test, mock exam or practice exam is done. Then based on the performance, set a new question paper accordingly. But this naive way is not efficient because the result of the student's performance will be known only after finishing the test, mock exam or practice question set. This process of selecting questions for the test, mock exam or practice question set from a question bank can be automated.

To solve this problem and improve efficiency of selecting the right questions, the selection of questions can be made dynamic. In dynamic question selection, the questions are selected or recommended while the student is attempting the test, mock exam or practice question set.

In this paper, we have proposed improvements and modifications to three existing reinforcement learning algorithms to build this dynamic personalised question recommendation which will help students to enjoy the learning process while still being challenged.

Keywords - multi-armed bandit; epsilon greedy; softmax; upper confidence bound(UCB); Thompson sampling; action; reward; action value; exploration; exploitation; N-class.

I. INTRODUCTION

Tests, mock exams and practice question sets give students an evaluation of the student's understanding in a subject. But they do not help the student to improve his / her existing knowledge in the subject directly. If a student wants to know the topics in which he / she is poor in and needs to improve, then he / she will come to know only after attempting the test, mock exam or practice question set and checking explicitly in exactly which topics he / she needs to improve. This process takes a lot of time and is not efficient. This process of making tests, mock exams and practice question set can be automated as questions can be selected from the question bank. In this paper we have proposed few methods to solve all of these problems using improvements and modifications in three reinforcement learning algorithms.

First lets get familiar with a general class of problems called multi-armed bandit problem. Briefly, multi armed bandit problem is a problem where there are certain choices to make and each choice has a certain reward. The goal of the problem is to select the choices to maximize the obtained rewards. To give an idea of why this problem is related to the smart question recommender, the choices can be considered to be the questions and based on the user's performance in the question, certain reward is given. Further details of multi-armed bandit problem and various algorithms is given in the next section.

However, smart question recommender would need to classify the topics into different classes like "easy", "medium", "hard" etc, to take decision on which question to recommend next. But traditional multi armed bandit problems do not take into account multiple classes for each choice. To overcome this problem, modifications and improvements on existing algorithms have been made to suit the requirements of the smart question recommender. The modified and improved algorithms have been tested and evaluations have been made to find the effectiveness of these algorithms.

II. LITERATURE REVIEW

Multi-armed bandit problem

The multi-armed bandit problem is a traditional problem where there are a number of different options (or actions) to choose from and each of these options (or actions) have a reward associated with it. The challenge here is to maximize the total reward earned over a sequence of choice of options (or actions).
The word "bandit" here refer to slot machines with an arm or lever associated with each of them. Activating any of these arms triggers the machine(bandit) and a reward associated with each of this is received. Here we come across a trade-off between exploration and exploitation.
If we focus more on exploring for finding the sequence with the maximum rewards associated with it (best possible solution for the problem), the more is the risk associated with it to lose more amount as we want to start exploiting our findings as early as possible to maximize our return.
On the other hand if we focus more on exploitation without exploring the given set of options, we may end up with a solution which is far from being the best possible solution. Thus, there is a "Regret" associated in this case which is a quantified measure of loss when non-optimal solutions is used.
The problem is to find the best way to maximize the rewards for the given set of probability distribution for rewards for the bunch of machines.

Here, we give a systematic review of the available solutions in the context of this problem and advantages some of these algorithms will have over others in applications having multi-armed bandit problem as its core.

Algorithms for multi-armed bandit problem

In the multi-armed bandit problem, the goal is to choose the best action among k different actions so as to maximize the expected total reward over some time period. Let $Q_n(a)$ denote the estimated value of the action at time step n. Then

$$Q_n(a) = \frac{r_1 + r_2 + r_3 + \ldots + r_{n_a}}{n_a}$$

where r_i denotes the reward yielded by action a at time i and n_a denotes the number of times action a is chosen prior to time step *n*. Let Q*(a) denote the true (actual) value of action *a*. If $n_a = 0$, then $Q_n(a)$ is set to some default value, say $Q_0(a) = 0$.

Some of the major algorithmic approaches to solve the multi-armed bandit problem are:

1) Epsilon-Greedy (ε - greedy):

The simplest approach is the greedy approach, i.e to select action *a* that has the highest estimated action value at time step *n*. $Q_n(a^*) = \max_a Q_n(a)$, where *a** is the greedy action chosen at time step *n* among all actions a. But this method does not explore other possible better actions as it only exploits the current knowledge to maximise immediate reward.
To include exploitation also, the algorithm can be modified such that it exploits most of the time but explores other actions with a small probability - ε .

This method is called ε - greedy method.

2) Upper Confidence Bound (UCB):

Rather than directly using $Q_n(a)$ to select the action at time step *n*, UCB introduces confidence interval for each of the k actions. As the number of particular action selected increases, the length of the confidence interval for that action decreases.
Let, $U_n(a) = Q_n(a) + \sqrt{\frac{2\,ln(n)}{n_a}}$
where, $Q_n(a)$ is estimated value of action *a* at time step *n* and n_a denotes the number of times action *a* is chosen prior to time step *n*.

Initially, all the actions are selected once.
Later, in each time step *n*, action a* is chosen among k actions such that $U_n(a^*) = \max_a U_n(a)$.
Length of confidence interval of a* decreases in that time step.

3) Thompson Sampling (Posterior Sampling):

In Thompson Sampling, initially, estimated value of each action, Q(a) is initialised as some probability distribution (eg: beta distribution).

In each time step, a sample is drawn from the distribution. Next, the action with the maximum value, a*, is selected among possible k actions. Next, actual reward for a* is found and probability distribution - Q(a*), is updated according to the actual reward of a*.

This process is repeated for each time step. After a considerable number of time steps, the probability distributions would converge to the true action values Q*(a). So when a sample is drawn from this converged distribution, the action selected would be that which has the greatest reward.

Brief analysis of the algorithms

ε - greedy algorithm is an improvement over the simple greedy algorithm. This is because ε - greedy algorithm allows room for exploration with probability ε along with exploitation, while simple greedy approach just uses exploitation.

Most of the applications of multi armed bandit problem use upper confidence bound algorithm(UCB). UCB is an easy algorithm to implement.

Recently there is increasing interest in Thompson Sampling algorithm to solve multi armed bandit problem. This is because results have shown that Thompson Sampling outperforms UCB algorithm. But there is still research going on in this area.

Other Applications

There are many areas where multi armed bandit problem is used. Most of the applications are for monetary purposes. This problem is also used as a base to solve optimization problems. It is used in the medical field too. Some major applications are given below.

1) Internet Advertising: Finding the most effective advertisement and that which will have the most impact among a number of possible advertisements. Here, in each time step, an advertisement is displayed and reward is gained if a user clicks on it.

2) Recommendation systems: The challenge of finding the right content(movie, news, posts) to recommend to a particular user. In each time step, a content is recommended to a user and reward is gained if the user clicks on it.

3) Network server selection: Finding the right server among a number of servers to process a request. The processing speeds may be different because of factors like load, distance etc. This has wide applications in cloud computing and routing.

4) Clinical trials: The problem where a doctor needs to choose the most effective treatment among a number of treatments to be administered to patients. Here, in each time step, a treatment must be selected and the reward is the survival or health of the patient.

Insights

In this systematic review we studied the various aspects of the different algorithms used to solve the multi-armed bandit problem which showed us three major insights :

- The parameters of heuristic algorithms like ε - greedy action selection can be changed to give better results for a given problem.
- The Upper Confidence Bound (UCB) is an easy algorithm to implement and is used in many applications.
- The Thompson Sampling algorithm is gaining more traction as results have shown that it gives better results than UCB.

These bandit algorithms find their use in a variety of applications like online advertising or network routing. Improvements in these algorithms will lead to a significant advancements in these practical uses cases.

III. PROBLEM DESCRIPTION

Student Specific Smart Question Recommender

When students study a subject or prepare for an exam, they often practice by solving questions from either a question bank or book. But these questions will be randomly selected most of the time and will belong to different topics. Each student has different strengths and weaknesses in different topics. So random selection would not be helpful for the students to learn as most of the time the questions selected are easy, in which case

the student doesn't learn much, while some the questions maybe very hard, in which case the student may be demotivated and may lose interest.

To solve these problems and issues, a personalised question recommender system is required to select the right questions, that is questions that are neither very easy nor very hard, to be recommended to the student. This will help the student learn the topic without being overwhelmed while still being challenged. This will make the learning process by solving questions a fun experience too.

IV. SOLUTION PROPOSED

Lets bridge the gap between the problem statement and multi-armed bandit problem. In the problem statement, the goal is to recommend questions from a topic to students that are neither too easy nor too hard so that the student feels confident as well as challenged. Basically, the algorithm should select the right topic from which questions can be recommended.

So, here the 'arms' are the topics. But each topic has different difficulty level for the user. These difficulty levels may be easy, medium, hard and many others. The traditional multi-armed bandit problem does not take into account multiple classes in a single arm. Here each difficulty level is a class. That is, 'easy' is one class, 'medium' is second class, 'hard' is third class and so on.

To solve the problem as given in the problem description, that is, to build a personalised question recommender, we have come up with a general class of improvement and modification in existing algorithms to solve multi-armed bandit problem. We have named this set of new algorithm as "**N-class multi-armed bandit solution**". Further details of this new algorithm is specified in the next section.

N-Class Multi-armed bandit solution

In traditional multi-armed bandit problem, there are K arms and each arm has a certain reward and there are only two classes - taken or not taken. But in the problem statement, each 'topic' of a subject is an arm. Moreover, each arm needs to divided into more than two classes. Here the classes can be 'easy', 'medium', 'hard' and so on.

To take into account, these classes, we have come up with improvements and modifications to few existing algorithms to solve the multi-armed bandit problem to make it a solution for the "**N-class multi-armed bandit problem**". If there are three classes - 'easy', 'medium' and 'hard', then 'N' in 'N-class' is equal to 3, that is N = 3 . The improvements and modifications made include assigning classes to each arm and assigning rewards to each class.

V. EXPERIMENTAL RESULTS AND ANALYSIS

The code for the algorithms is written in python. This code simulates the actual working of the student specific question recommender if the algorithms were to be used in an application. To simulate the results, a test dataset was created so as to show the performance and effectiveness of the algorithms. A dataset would not be required in actual deployment as these are reinforcement learning based algorithms and the algorithms learn from the actions of the users over time.

The test data created was meant to simulate a situation where there are 10 topics and each topic has 80 questions. Each question can be classified into 3 classes - "easy", "medium" and "hard". These topic can be classified into these 3 classes based on the time taken by the student to solve questions from these topics. Each of these classes is also given a certain reward. For the current scenario, "easy" and "hard" have a reward of 0 and "medium" has a reward of 1. The main idea in assigning these rewards is to prevent users from solving "easy" questions from topics in which they are already comfortable in, to not overwhelm the user with too many difficult questions as the user may feel discouraged. So more emphasis is given on "medium" class of questions so that the user will be challenged and not be discouraged at the same time.

The test data was created according to the following details:

Topic	Number of questions per 80 questions the user felt as easy, medium or difficult		
	Easy	Medium	Hard

Topic A	40	25	15
Topic B	20	27	33
Topic C	24	32	24
Topic D	12	51	17
Topic E	38	24	18
Topic F	43	21	16
Topic G	19	48	13
Topic H	34	29	17
Topic I	15	18	47
Topic J	41	19	20

From the above table we see that, topics A, E, F and J were "easy" for the user. Topics B and I were "hard" for the user. Topics D and G were of "medium" difficulty. So according to the proposed plan, topics D and G must be recommended most of the time as they have high percentage of questions which the user found to be of "medium" level.

Next sections will cover the simulation results and insights of the algorithms proposed.

Random Selection

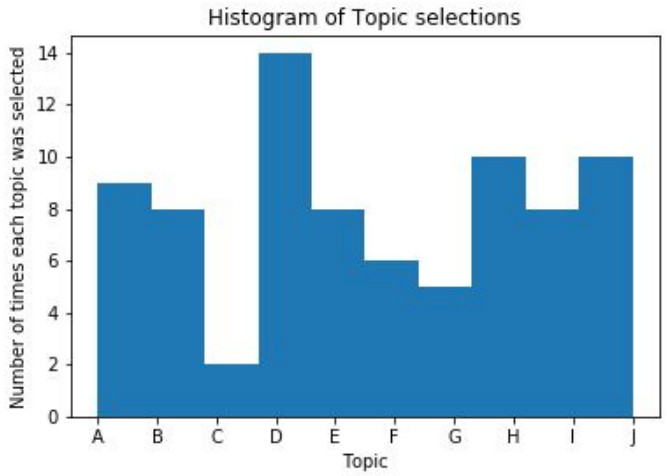
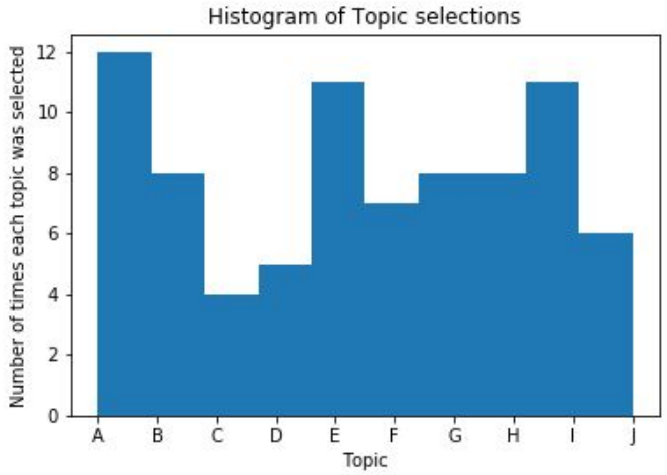
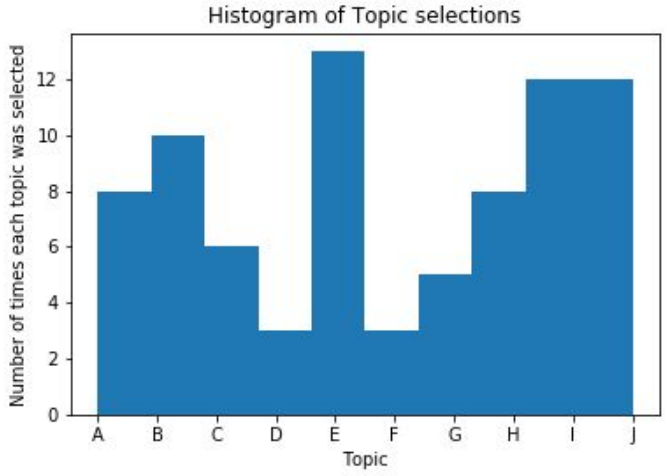
Many applications that provide test, mock exams, quiz, practice papers randomly select the question to be shown to the user for them to solve. There are many ways to select questions randomly from a database. These questions maybe selected on the basis of a pseudo random number generator or round robin method. Selecting questions randomly is the bare minimum functionality of a quiz application as it satisfies the basic need of selecting questions from a database for the user to solve. If a developer wants to just build a basic quiz application, then random selection would satisfy most of the requirements.

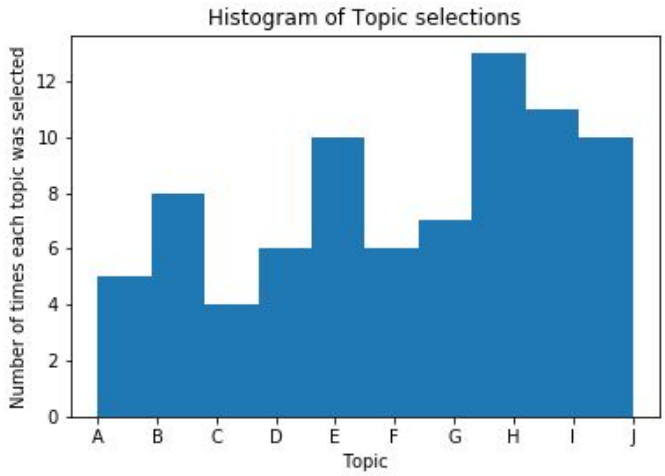
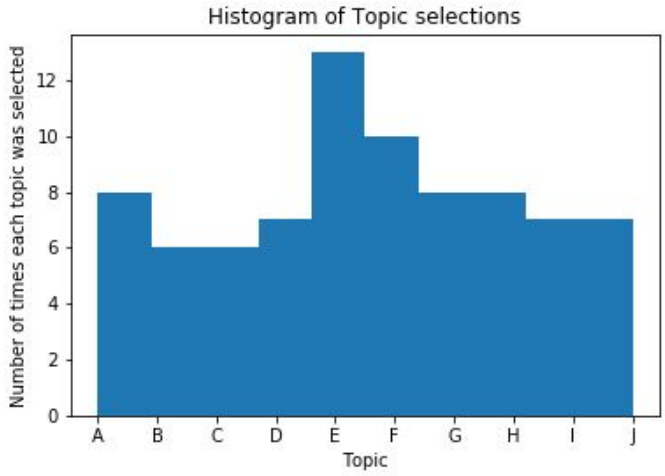
Random selection of questions from a certain topic is also probably how most of the exam papers for schools, colleges or entrance exams are created. While preparing for exams, sometimes students select questions from a textbook randomly and try to solve them. Random selection of questions seems to be the most used method of questions selection whether it is done knowingly or unknowingly. But the most widespread method may not always be a good solution. Random selection may be the most widespread method because there may be only a few other methods which performs of the job of selecting the questions from database.

Random selection does not give insights as to why a particular question needs to be chosen. There is no decision made as to what question to choose as selection is done randomly. Hence there is always room for improvement and come up with better selection or recommendation algorithms to solve specific use cases. Especially, in the case where students need personalised recommendations of questions based on their strengths and weaknesses.

Since random selection is widely used, random selection is taken as the base case or benchmark. The following table displays the results of simulation of random selection of questions. Total reward is calculated by summing over the rewards of each question based on user's performance in the question.

Random selection		
Sl no	Total reward	Histogram (Random selection)

1	25	 Histogram of Topic selections Number of times each topic was selected 0 2 4 6 8 10 12 14 A B C D E F G H I J Topic
2	27	 Histogram of Topic selections Number of times each topic was selected 0 2 4 6 8 10 12 A B C D E F G H I J Topic
3	22	 Histogram of Topic selections Number of times each topic was selected 0 2 4 6 8 10 12 A B C D E F G H I J Topic

4	23	
5	27	

Average total reward : (25+27+22+23+27) / 5 = 24.8

Maximum reward : 80

From the above table, we observe that random selection is not performing that well as the total reward low. This is the expected behaviour as random selection has no special properties to make question selection

1. **N - class Epsilon-Greedy (ε - greedy):**

To improve over the naive approach of random selection and give better results through reinforcement learning, modifications and improvement over the epsilon - greedy algorithm is made to fulfil the requirements of the problem's solution. The epsilon - greedy algorithm is one of the simplest algorithm yet, a powerful one if customized for given application.

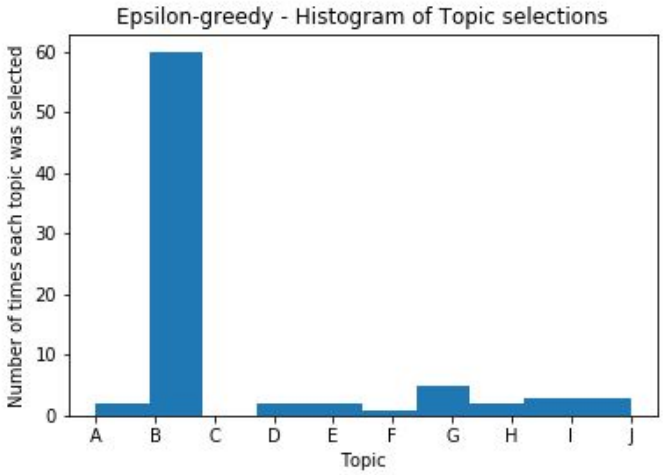
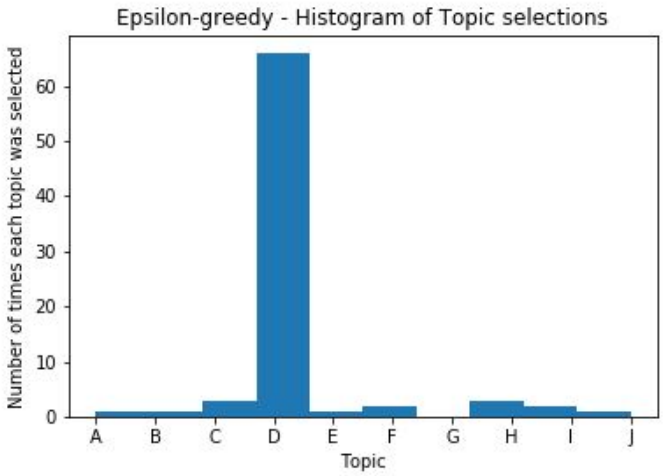
Traditional epsilon - greedy algorithm considers arms to be of single class. But the problem statement proposed requires that each arm, that is the topics, be multi-class or N-class. These classes are "easy", "medium" and "hard" as assumed in the simulation. Each of these classes have a reward. "Easy" and "hard" have reward of 0 and "medium" has a reward of 1. So to solve the N-class multi-armed bandit problem, following is the improved and modified algorithm.

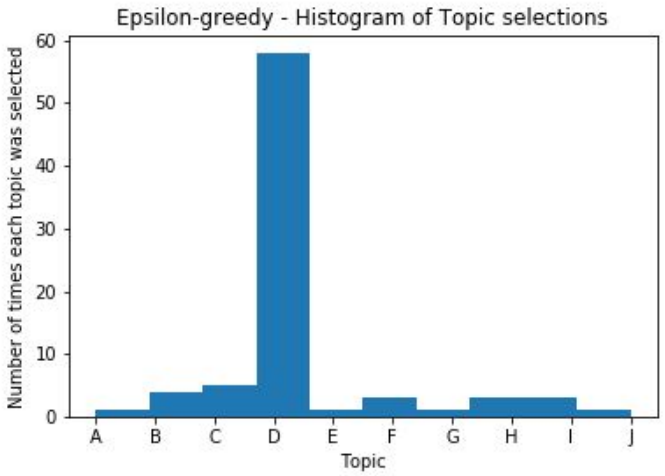
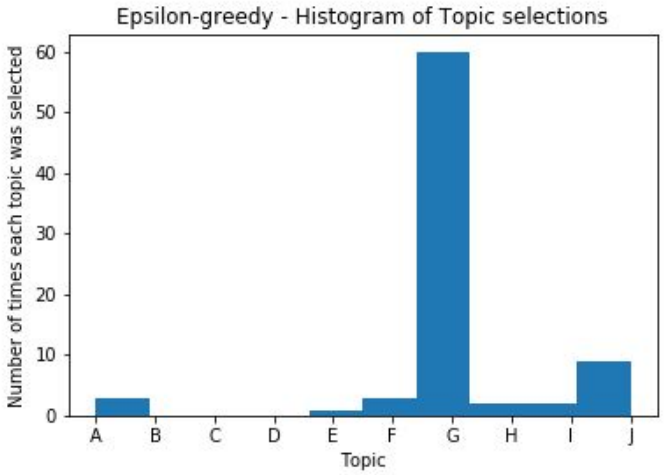
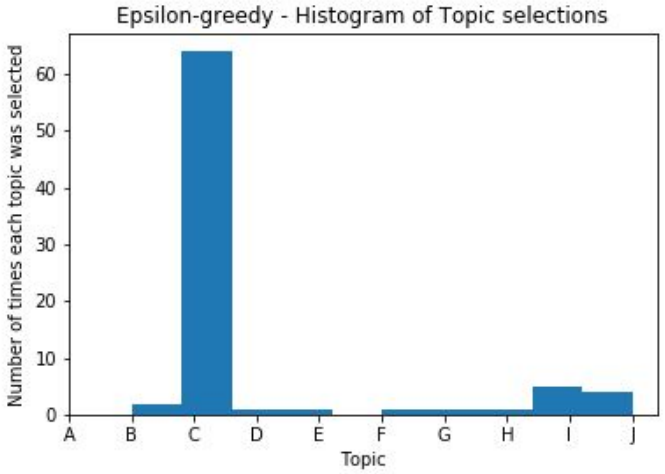
N - class Epsilon - Greedy algorithm for smart question recommender

1. Set an initial epsilon value to be of 1. This epsilon value is to determine the probability that the algorithm will choose to explore or exploit.

2. Set a gamma value to be about 0.95. This gamma value is the discount factor which needs to be multiplied to epsilon in every iteration. This discount factor is useful to reduce the probability of exploration and increase the probability of exploitation as time proceeds.
3. In every iteration, choose a random real number between 0 and 1.
4. If the random number is less than the epsilon value, then choose a random topic and questions from that topic will be displayed to the user.
5. If the random number is more than epsilon value, then choose the topic which has the highest reward until now. Question from the chosen topic will be selected to be displayed to the user.
6. Once the topic is selected, questions from that topic will be displayed to the user.
7. The class (easy, medium or hard) to which the question from that topic must belong, will be decided based on the user's performance while solving the question. If a user takes less time, say less than 3 min, then the question is easy. Else if the user takes a lot of time or is unable to solve the question, then the question from that topic is difficult. Else the question is of medium level.
8. Once the class of the question is decided, the reward for that class will be added to the total reward. Also the sum of reward for particular topic will be stored and will be useful while finding the topic with the highest reward.

Results of simulation of this algorithm is shown below. The results obtained will be different even for the same scenario as the trade off between exploration and exploitation is randomized. So five different results are shown below.

N - class Epsilon-Greedy		
Sl no	Total reward	Histogram (N-class epsilon greedy)
1	31	
2	47	

3	43	
4	46	
5	33	

Average total reward : (31+47+43+46+33) / 5 = 40

Maximum reward : 80

From the above table, you can observe that the most of the time the algorithm is able to recommend questions from topics D and G which of medium level. This is exactly what the desired result is. It also gives a very good reward compared to random selection.

However, in certain cases, it selects topics other than D and G with a large frequency. The graph 1 selects topic B and graph 5 selects topic C with high frequency due to which they have a much lesser reward compared to other graphs. This is one of the drawbacks of epsilon - greedy algorithm. It needs the right critical value for exploration and exploitation for a given use case. With the right values for parameters like epsilon, gamma, this algorithm can be made powerful.

2. N - class Upper Confidence Bound (UCB):

Upper Confidence Bound (UCB) algorithm overcomes some of the shortcomings of epsilon - gradient algorithm. Traditional Upper Confidence Bound (UCB) algorithm is easy to implement. Converting this UCB into N-class UCB is required for smart question recommendation as each arm, that is the topics, have different difficulty level for that particular user. The N-class Upper Confidence Bound is follows.

N-class Upper Confidence Bound algorithm for smart question recommender

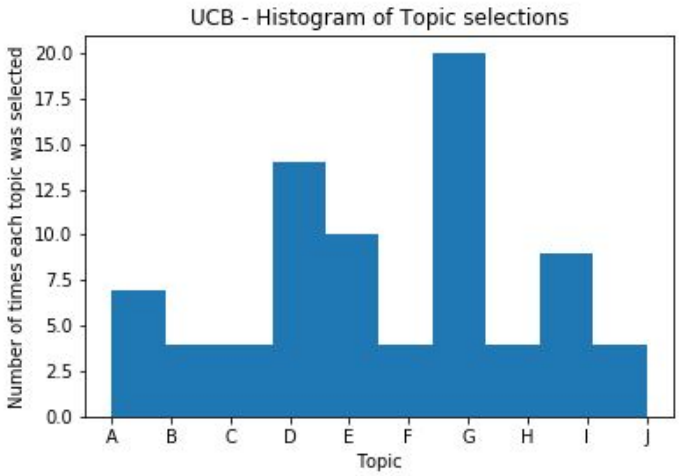
1. Initialise the total reward, sum of rewards of each topic, number of selections of each topic to zero.
2. In every iteration, the topic with the greatest upper confidence bound is chosen
3. To ensure that all the topics are displayed to the user at least once, the algorithm checks the number of selections for each topic. If a topic has number of selections as zero, then that topic needs to be selected. To select that topic, the upper confidence bound for that topic is set to a large to ensure that it will be selected.
4. If all topics are selected at least once, then the upper confidence bound is calculated for each topic. The calculation for upper confidence bound is:

Upper confidence bound =
average reward of topic until then +

$$\sqrt{\frac{3}{2}\frac{log(n)}{number\ of\ times\ questions\ from\ that\ topic\ was\ "medium"}}$$

5. The topic with the greatest upper confidence bound value is chosen and its question to be displayed.
7. The questions from the chosen topic is displayed to the user
8. Based on the user's action on question from the chosen topic, the class of the topic is identified, that is easy, medium or hard.
9. Once the class is identified, the reward for that class is used to update total rewards and sum of rewards for each topic. These updates will be used in next iterations for important calculations.

Unlike the random selection and epsilon - gradient, N-class upper confidence bound (UCB) has no random component to it. So it gives the same results for that scenario. Hence only one graph has been shown.

N - class Upper Confidence Bound (UCB)	
Total reward	Histogram (N-class UCB)
36	

Average total reward : 36

Maximum reward : 80

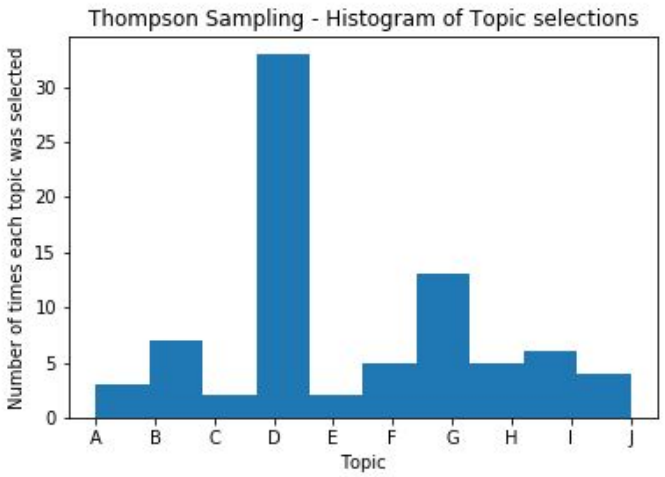
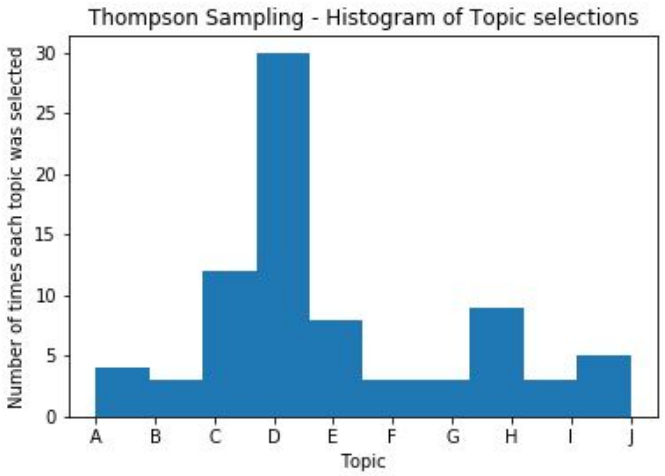
This N - class Upper Confidence Bound (UCB) algorithm definitely gives better results than random selection. Although N - class Upper Confidence Bound (UCB) algorithm gives lesser reward than epsilon - greedy algorithm in this case, it is to be noted that UCB is more reliable than epsilon - greedy. This reliability can be examined by comparing the graph plots of N-class UCB and N-class epsilon - gradient. Topics D and G have higher frequencies that other topics in N - class UCB always. However, this level of reliability was not guaranteed in N-class epsilon - gradient.

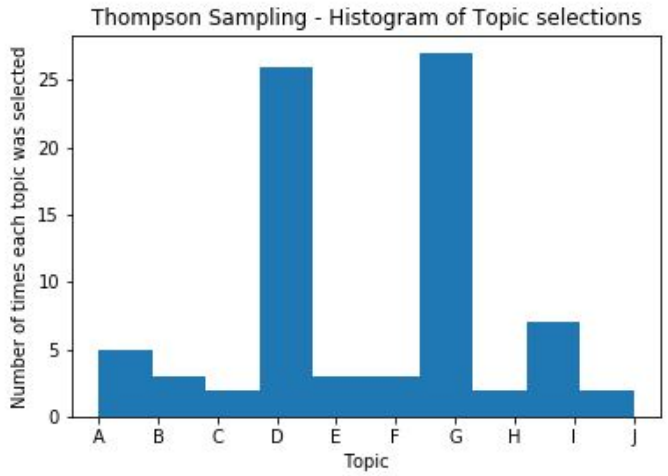
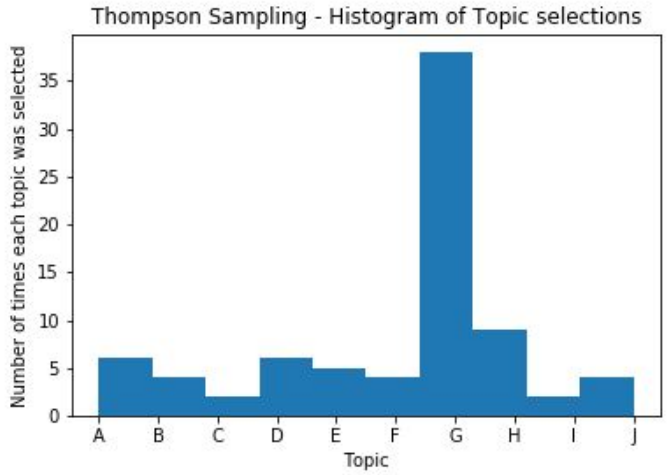
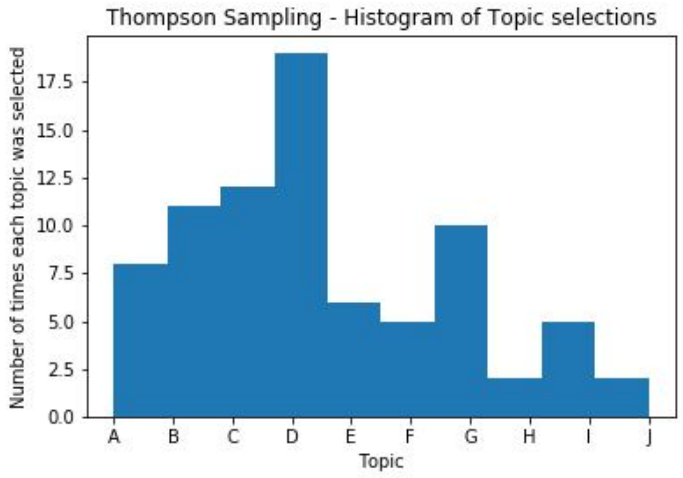
3. N - class Thompson Sampling

Thompson's sampling is said to perform better than most algorithms. Because of this it is gaining traction in the present day. This also started being used very recently in the industry. A lot of research is also going on in this area as the results show that this performs better than UCB. Though this algorithm has a little bit of randomness in it, unlike the epsilon - gradient algorithm, Thompson's sampling is quiet stable and gives almost similar results in each try. Traditional Thompson's sampling algorithm does not incorporate multiple classes in each arm. So this algorithm can be modified to take into account multiple classes(N-class) in each arm to build the smart question recommender. The N-class Thompson Sampling algorithm is as follows.

N-class Thompson Sampling algorithm for smart question recommender

1. Initialise variables - total reward, (number of times a topic got reward 1) and (number of times a topic got reward 0) to zero
   In each iteration,
2. For each topic, the number is sampled from the beta distribution with alpha value being 1 plus the number of times the topic got reward 1 and beta value being 1 plus the number of times the topic got reward 0.
3. The topic which has the maximum value of sampled number is selected to be displayed to the user.
4. Based on whether the user found the question "easy", "medium" or "hard", the total reward value is updated. Also the variables that store the count of number of times a topic got reward 1 and number of times a topic got reward 0 is updated.
5. This process with the updated variables and total reward is used in future iterations.

N - class Thompson Sampling		
Sl no	Total reward	Histogram (N-class Thompson Sampling)
1	37	
2	35	

3	39	
4	38	
5	34	

Average total reward : (37+35+39+38+34)/5 = 36.6

Maximum reward : 80

From the table of results, we see that N-class Thompson's Sampling, gives similar rewards in different tries, unlike the N-class epsilon - gradient algorithm. Although the final output graphs look different look different, the algorithm selects topic D, topic G and even both most of the times. This is exactly what is to be expected given the current scenario (test data) as the user found questions from these topics to be of "medium " type most of the time. It is also performs a little better than N-class UCB algorithm in this scenario.

V. CONCLUSION

Traditional multi armed bandit algorithms assume each arm to be of a single class. But in the context of smart question recommender, each arm (topic) should have multiple classes. These classes are divided so as to find and classify user's strengths and weaknesses in different topics. These classes can be "easy", "medium", "hard" etc. Traditional multi armed bandit algorithms do not incorporate multiple classes in each arm. Hence there is a need to modify and improve the algorithms to take into account multiple classes in each arm. These improved algorithms can help solve the problem of smart question recommender.

Three algorithms that solve the multi armed bandit problem, which are epsilon - greedy, upper confidence bound and Thompson's sampling were improved and modified to take into account multiple classes ("easy", "medium", "hard") in each arm (topic). These modified and improved algorithms were given the name of N - class epsilon - gradient algorithm, N - class Upper Confidence Bound (UCB) algorithm and N - class Thompson Sampling algorithm. These algorithms were tested using a pre - built test data set to simulate a scenario where there are 10 topics A-J and a user finds most of questions in topics D and G to be of "medium" type while others had most of them to be of "easy" or "hard".

Random selection was considered to be the benchmark solution as that is the algorithm used in most of the practice quiz, mock test applications. Compared to random selection, N - class epsilon - gradient algorithm, N - class UCB algorithm and N - class Thompson Sampling algorithm performed much better as they selected the topics D and G most of the time, which is exactly what was expected.

There is always room for improvement. Some possible improvement that can be made to the algorithms to build the smart question recommender is to have variable rewards in each class in different arms of the multi armed bandit problem. There is always a tradeoff between exploration and exploitation. Finding algorithms that reduce the time required for exploration so that exploitation can be given more importance is also another challenge.

VI. REFERENCES

N. Gupta, O. Granmo and A. Agrawala, "Thompson Sampling for Dynamic Multi-armed Bandits", 2011 10th International Conference on Machine Learning and Applications and Workshops, 2011.

C. Tekin and M. Liu, "Online algorithms for the multi-armed bandit problem with Markovian rewards", 2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton), 2010.

P. Auer and R. Ortner, "UCB revisited: Improved regret bounds for the stochastic multi-armed bandit problem", Periodica Mathematica Hungarica, vol. 61, no. 1-2, pp. 55-65, 2010.

D. Koulouriotis and A. Xanthopoulos, "Reinforcement learning and evolutionary algorithms for non-stationary multi-armed bandit problems", Applied Mathematics and Computation, vol. 196, no. 2, pp. 913-922, 2008.

A. Mersereau, P. Rusmevichientong and J. Tsitsiklis, "A Structured Multiarmed Bandit Problem and the Greedy Policy", IEEE Transactions on Automatic Control, vol. 54, no. 12, pp. 2787-2802, 2009.

E. Kaufmann, N. Korda and R. Munos, "Thompson Sampling: An Asymptotically Optimal Finite-Time Analysis", Lecture Notes in Computer Science, pp. 199-213, 2012.

K. Jamieson and R. Nowak, "Best-arm identification algorithms for multi-armed bandits in the fixed confidence setting", 2014 48th Annual Conference on Information Sciences and Systems (CISS), 2014.

K. Huang and H. Lin, "Linear Upper Confidence Bound Algorithm for Contextual Bandit Problem with Piled Rewards", Advances in Knowledge Discovery and Data Mining, pp. 143-155, 2016.

S. Raja, "Multi Armed Bandits and Exploration Strategies", Sudeepraja.github.io, 2018. [Online]. Available: https://sudeepraja.github.io/Bandits/

"Solving the Multi-Armed Bandit Problem – Towards Data Science", Towards Data Science, 2018. [Online]. Available: https://towardsdatascience.com/solving-the-multi-armed-bandit-problem-b72de40db97c.

"Multi-Armed Bandit", Optimizely.com, 2018. [Online]. Available: https://www.optimizely.com/optimization-glossary/multi-armed-bandit/

"The Epsilon-Greedy Algorithm", James D. McCaffrey, 2018. [Online]. Available: https://jamesmccaffrey.wordpress.com/2017/11/30/the-epsilon-greedy-algorithm/

"The Upper Confidence Bound Algorithm", Bandit Algorithms, 2018. [Online]. Available: http://banditalgs.com/2016/09/18/the-upper-confidence-bound-algorithm/

R. Sutton and A. Barto, Reinforcement learning. Cambridge, Massachusetts: The MIT Press, 2012.