Review of Multi-Arm Bandit Algorithms in Recommender Systems

Amith Nair

[1] Trinity College Dublin, Ireland
[2] nairam@tcd.ie

Abstract. In a fast moving dynamic world with new information being inserted into the internet on a per second basis, recommendation systems need to evolve so that the recommendations given by it are still valid. In order for this the learning agent or the recommendation system needs to learn more about the user also making sure that the user is satisfied. Achieving the right balance between exploitation and exploration thus becomes extremely crucial. One of the most popular ways to model this problem is the multi-arm bandit problem. This paper reviews some of the widely used algorithms to solve the multi-arm bandit problems right from the most simplest algorithms like epsilon greedy to state of the art and more complex algorithms like the contextual bandit algorithms.

Keywords: Bandit Algorithms · learning agent · sampling.

1 Introduction

The plethora of information in the internet has made recommendation systems extremely relevant in today's world. For a user, a simple search functionality gives him/her a lot of irrelevant information. Most of this irrelevant information can be filtered out with the help of a recommendation system. A popular method of recommendation system is the Collaborative Filtering model. This model takes into account the user item interactions and suggests items which are used by similar users. Other than providing recommendations based on the information already available, in a dynamic world it is also necessary to learn new information about the user. Hence some uncertain recommendations should also be made to explore more about the user. However, there needs to be a right balance between exploration(satisfying users through useful recommendations) and exploitation(knowing more about the user). This exploration-exploitation trade-off can be modeled in to a multi-armed bandit problem. We will first discuss the Multi-Armed Bandit Problem followed by a review of the some of the algorithms used to solve this problem

1.1 Multi-Armed Bandit problem

Consider a slot machine with n arms. Each of these arms when selected gives a random reward between 0 and 1. One of these n arms need to be selected

at each iteration. The goal is to maximize the cumulative rewards obtained. The reward is unknown to the agent(the recommendation model) before the selection of the arm and is provided to the agent right after selecting the arm. The random reward distribution of any of the arm are i.i.d variables. The n arms represents the options that could be recommended to the user and the selected arm represents the recommendation given to the user. The agent is rewarded based on whether the user likes or clicks on the recommendations.

To maximize the cumulative rewards obtained the Multi-armed Bandit agent will need to select the arm which gives the maximum reward at every step. The agent makes this decision based on the information it has gained through the last iterations. One way to model this problem is by maximizing the expected total reward

$$R(T) = E[\sum_{t=1}^{t=T} u_i(t)]$$

where $u_i(t)$ represents the reward obtained by selecting arm i. Most commonly, the problem is represented in the form of regret. Regret represents the difference between the cumulative rewards that could have been achieved by selecting the optimum policy every time and the rewards obtained so far.

If the agent only relies on the information it has gained so far, it will only select the arm which appears to be optimum at that step. However in the long term it could be a sub optimum solution. Hence the best long term strategy is to balance the exploration and exploitation by making short term sacrifices.

2 Multi-Armed Bandit Algorithms

There are many algorithms to solve the multi-armed bandit problem. Some of them are discussed below:

2.1 Epsilon Greedy

At any iteration, this algorithm selects the arm which is known to give the maximum reward with a probability of $1 - \epsilon$. It takes a random action with a probability of ϵ. The ϵ factor tries to balance exploration and exploitation. The regret at every iteration by this method is linear in nature due to the constant epsilon factor. In the paper [8], an approach to solve the cold start problem in recommendation engines has been proposed which uses an epsilon greedy algorithm along with k means clustering. In cases where the environment is not constantly changing, this algorithm has a drawback that it keeps on exploring forever. In such a case, the exploring factor should decrease once enough information of the user is known. By decaying ϵ at each iteration, the algorithm can be modified to explore and learn more about the user at the start. Once enough information about the user is gathered, the leaning agent can reduce the exploration factor and exploit more. This method is called the decaying ϵ greedy algorithm.

2.2 Upper Confidence Bound(UCB)

This algorithm takes into account the mean reward obtained for selecting any of the arms along with its corresponding confidence interval. The upper confidence bound value for each of these arms is calculated and the arm which gives the highest upper confidence bound value is then selected. UCB has a logarithmic reward function. Many variations in the UCB value calculations have been discussed. The first iteration of UCB[2] developed selects the arm which maximizes

$$\overline{x}_j + sqrt(2 * \ln(\frac{n}{n_j}))$$

[2].

x_j represents the reward that could be obtained based on previous information and the second factor is the upper confidence value.

A slight variation in the UCB1 algorithm replaces the upper confidence value with

$$\sqrt{(}\frac{\log(t)}{N_t(a)} min(1/4, (\sigma_a + 2\frac{\log(t)}{N_t(a)})))$$

[2]

σ_a is the sample variance for each action a. This variation called the UCB1-tuned [2] performs much better than UCB1.

Another variation of the UCB called the UCB2 is discussed in [2] in which the same set of recommendations are shown to the users for a while. Once all the users provide their feedback , the agent learns from the feedbacks. It was shown that UCB 2 performs slightly worse than UCB1. Paper [9] has designed an algorithm that runs UCB2 in parallel to provide a small set of recommendations from a huge dataset. It was seen that after tuning UCB2, the recommendations given by UCB2 were on par with that given by UCB1.

2.3 Thompson Sampling

Paper [10] puts forward a Bayesian approach to minimize the regret. For each of the arms of a multi-armed bandit, a Beta prior distribution is assumed about the mean of the reward distribution. An arm is selected based on the posterior distribution which tells us the most rewarding arm. After every iteration, the posterior is updated based on the information obtained in that iteration.

Initially all the arms are assumed to have a prior distribution of $\beta(1, 1)$. This assumes that all of the arms give a mean reward of 0.5. The posterior distribution for every arm is then updated every time the arm is selected as $\beta(n_s + 1, n_f + 1)$ where n_s is the number of times a reward was obtained and n_f is the number of times no reward was obtained.

[4] evaluated the Thompson Sampling method on some real world recommendation systems like news recommendation and advertisement selection. It was seen that Thompson sampling achieved state of the art results and in some use cases surpassed the results obtained by UCB. Thompson Sampling is used in Microsoft's Ad Predictor[6]

2.4 Online Clustering of Bandits

When the number of users and items is huge, the computations involved in the recommendation process increases significantly. Paper [5] introduces a recommendation algorithm based on adaptive clustering of bandits. The algorithm tends to find relationships between various users and form groups of users. The content is then served based on the common interests of the users.

The algorithm starts with a complete graph where all the users are clubbed into one huge cluster and every user is linked to every other user. There is a bandit at the user level as well as the cluster level. The users are provided recommendations based on the cluster in which they are present. At each iteration, both the user and the cluster updates its estimate of the reward value associated with each arm. When the difference between the estimates of two users become more than a certain value, the link between them is deleted and the clusters are recalculated. After some iterations we obtain the true clusters of users which effectively group the users based on their interests. This algorithm was tested on some synthetic and real world datasets like the Yahoo datasets. The clustering algorithm performs significantly better than most of the currently used baseline algorithms.

The algorithm introduced by paper [5] has two major drawbacks. Firstly, it assumes that users will be served in a uniform fashion which ensures that the interests of all the users will be updated uniformly. However in the real world, the users will not be served in a uniform fashion and there will always be some infrequent users. These infrequent users will cause discrepancies in the cluster that they are present in, thus lowering the performance of the algorithm. Secondly, it takes a long amount of time for new clusters to form since the link between any particular user and all other users in the cluster need to be deleted for that user to be removed from the cluster. Thus a user can remain in a cluster in which it does not belong to for a long time.

Paper [7] introduces an updated version of the online clustering algorithm in which the frequency of the user are also taken into consideration. This solves the first issue mentioned of [5]. To mitigate the second issue, the updated clustering algorithm separates a user from a cluster as soon as it is discovered that the cluster contains dissimilar users to this one. The algorithm adds a new operation of merging between the users to accelerate the process of finding the right cluster for users. Experiments on various synthetic and real world datasets show that this algorithm performs better than the clustering algorithm mentioned in [5]

2.5 Contextual Bandit Algorithms

In this algorithm, the agent before choosing an arm of the bandit sees feature vectors know as context. The agent uses this features and the rewards obtained in the previous iterations to make a decision about which arm to choose. The agent tries to learn how the context and rewards obtained on selecting each arm correspond to each other. The context vector represents the state of the environment. Example: For a news recommendation app, the context could be

the location, age or profession of the user. Paper [1] introduces a contextual Bandit algorithm coupled with Thompson Sampling. The regret bound achieved by these algorithm is best among all the algorithms belonging to the Bandit family.

The use of context in the bandit algorithms introduce a new problem call the Contextual Bandit with Corrupted Context[3]. This takes into account the possibility of the context information being corrupted. In this case for standard contextual bandit algorithms, the learning is skipped in such an iteration. [3] proposes a Thompson sampling based solution in which the algorithm reverts back to a normal bandit algorithm for the iteration in which a corrupt context is encounter which enables the agent to keep learning.

3 Conclusion

In this paper, we have discussed various types of multi-armed bandit algorithms. The complexity and efficiency of each of these algorithms vary. We also looked at some of the real world use cases of these algorithms. Some of the algorithms like the Contextual Bandit Algorithms are relatively new and developments are still going on in this field. Just like how a learning agent must adapt to each user based on new information, recommendation algorithms also need to improve to meet the ever growing data and demands of the user.

References

1. Agrawal, S., Goyal, N.: Thompson sampling for contextual bandits with linear payoffs. In: International conference on machine learning. pp. 127–135. PMLR (2013)
2. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine learning **47**(2), 235–256 (2002)
3. Bouneffouf, D.: Corrupted contextual bandits: Online learning with corrupted context. In: ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 3145–3149. IEEE (2021)
4. Chapelle, O., Li, L.: An empirical evaluation of thompson sampling. Advances in neural information processing systems **24** (2011)
5. Gentile, C., Li, S., Zappella, G.: Online clustering of bandits. In: International Conference on Machine Learning. pp. 757–765. PMLR (2014)
6. Graepel, T., Candela, J.Q., Borchert, T., Herbrich, R.: Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. Omnipress (2010)
7. Li, S., Chen, W., Leung, K.S.: Improved algorithm on online clustering of bandits. arXiv preprint arXiv:1902.09162 (2019)
8. Najem, Y.A., Hadi, A.S., Hamid, Z., Khafaji, H.K., Talal, R., Raouf, A., Mahmood, S.K., Sadeq, H., Abdulrahman, M.M., Abood, A.D., et al.: Aicis 2020
9. Rahman, M., Oh, J.C.: Parallel and synchronized ucb2 for online recommendation systems. In: 2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT). vol. 1, pp. 413–416. IEEE (2015)
10. Thompson, W.R.: On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometrika **25**(3-4), 285–294 (1933)