Application and Comparison of LinUCB and DivLinUCB in the News Recommending System

Aakash Roy | 21i190007
Srija Mukherjee | 21i190013

Guide: Prof. M. K. Hanawal

Industrial Engineering and Operations Research
Indian Institute of Technology, Bombay

Spring, 2022

Contents

1 Abstract

In real life scenarios, multi armed bandits help in making smart choices in highly dynamical settings where the pool of available options is rapidly changing and the set of actions to choose has a limited lifespan. However, adding auxiliary information about the user in this set up leads to better decision among all actions in MAB setting with the objective of maximizing the number of clicks. Here, we first implement the LinUCB algorithm to find out whether it is effective in building effective recommendation engines. However, focusing solely on maximizing click through rate may overexpose some articles whereas the remaining ones may be recommended very rarely. In this case it is rational to consider the "historical frequency", or the number of times it has been recommended as the "cost" of recommending it. This project is a comparison of performance between the above mentioned algorithms.

2 Introduction

In the modern day world, it is very important to understand the user, and based on his taste, come up with more personalised recommendations. With the help of a relevant user data stream, contextual bandits rely on an incoming stream of user context data, either historical or fresh, which can be used to make better algorithmic decisions in real time.

While the classic approach requires manual intervention once a statistically significant winner is found, bandit algorithms learn as you test, ensuring dynamic allocation of traffic for each variation. Contextual bandit algorithms change the population exposed to each variation to maximize variation performance, so there is not a single winning variation. Similarly, there is not even a single losing variation. At the core of it all is the notion that each person may react differently to different content.

In a contextual bandit problem, in each discrete trial $t = 1, 2, \ldots$, the model observes user u_t , a set of actions A_t and $f_{t,a}$ is the contextual information. The optimal arm selection strategy of the algorithm in the next round is based on the observation $(f_{t,a_t}, a_t, r_{t,a_t})$. Thus in a cumulative way, the total payoff till round T is given by $G_T = \Sigma_{t=1}^{T} r_{t,a_t}$. Similarly, we define the optimal expected total T trial payoff as $G_T^* = \Sigma_{t=1}^{T} r_{t,a_t^*}$, where a_t^* is the arm with maximum expected payoff at trial t. The **Regret** is defined as $R_T = G_T^* - E[G_T] = E[\Sigma_{t=1}^{T} r_{t,a_t^*}] - E[\Sigma_{t=1}^{T} r_{t,a_t}]$ The general goal of the multi-armed contextual-bandit problem is to design an algorithm that maximizes the expected total payoff G_T, which can be done by LinUCB or Contextual Thompson Sampling, etc. We assume that the expected payoff for an arm a depends linearly on the contexts, i.e., $E[r_{t,a}|f_{t,a}] = f_{t,a}^{\top}\theta_a^*$

In the multi-armed bandit setting, we use the UCB algorithm, which is also phrased as "optimism in the face of uncertainty". UCB does not depend on a time horizon and selects the optimal arm with high probability. So, in the contextual bandit problem also, we implement similar idea as the LinUCB algorithm. All the existing algorithm focuses more on selecting the optimal arm only. Thus, in news recommendation engines, some articles will be more preferred and will be highly over exposed. On the other hand, articles which are not of topmost preference will not be recommended at all. The DivLinUCB gives a tradeoff between click through rate and diversity.

3 Algorithms

In this section we will discuss the two above mentioned algorithms in greater details.

3.1 Linear UCB

For discrete time $t = 1, 2, \ldots$ Consider a set of K arms A_t. α is a tuning parameter, greater than zero. For each trial, we observe the features for all arms a as $x_{t,a}$. If the arm is being selected for the first time, we take A_a as $d-$ dimensional Identity matrix and the associated b_a vector as $d-$ dimensional zero- vector. A_a denotes the context matrix for arm a and b_a denotes the corresponding reward vector.$\hat{\theta}_a$ is the coefficient vector and $p_{t,a}$ is the Upper Confidence Bound for arm a.

Algorithm 1 LinUCB with disjoint linear models.

```
Inputs: α ∈ ℝ+
for t = 1, 2, 3, ..., T do
    Observe features of all arms a ∈ A_t: x_{t,a} ∈ ℝ^d
    for all a ∈ A_t do
        if a is new then
            A_a ← I_d (d-dimensional identity matrix)
            b_a ← 0_{d×1} (d-dimensional zero vector)
        end if
        θ̂_a ← A_a^{-1} b_a
        p_{t,a} ← θ̂_a^⊤ x_{t,a} + α sqrt(x_{t,a}^⊤ A_a^{-1} x_{t,a})
    end for
    Choose arm a_t = arg max_{a∈A_t} p_{t,a} with ties broken arbitrarily, and observe a real-valued payoff r_t
    A_{a_t} ← A_{a_t} + x_{t,a_t} x_{t,a_t}^⊤
    b_{a_t} ← b_{a_t} + r_t x_{t,a_t}
end for
```

The optimal arm a_t^* is selected as $argmax_{a \in A_t}\ p_{t,a}$, with ties broken randomly. The real valued reward r_t is observed at the end of every round. Consequently, the A_a matrix and b_a vector is also updated.

In order to evaluate the accuracy of this algorithm, we compute the Cumulative Take-Rate (CTR) value which is defined by

$$
\begin{aligned}
C_T &= \frac{Reward * \ Number\ of\ Clicks\ of\ arm\ a}{Number\ of\ recommendations} \\
&= \frac{\Sigma_{t=1}^{T} r_t \times 1[\pi_{t-1}(x_t) = a_t]}{\Sigma_{t=1}^{T} 1[\pi_{t-1}(x_t)}
\end{aligned}
$$

, where r_t is the reward generated at round t and π is the policy.

3.2 Diversity Boosting Linear UCB

Previously, we have already assumed that expected payoff depends linearly on context, and cost of choosing an arm can be incorporated in the payoff. In this new algorithm, cost is incorporated as $E[r_{t,a}|f_{t,a}] = \frac{f_{t,a}^{\top} \theta_a^*}{cost_{t,a}}$ and $cost_{t,a} = 1 + \beta(\ \frac{rc_a}{\hat{T}})^{\ n}$ β is a tuning parameter, greater than 0 and $cost_{t,a}$ implies the cost of selecting arm a in the t^{th} round.

Algorithm 2: DivLinUCB

Input: α, β, T time instances and at each instance i: set of arms available A_i, the displayed arm d_i, reward corresponding to it r_i and feature vector for each arm $a \in A_i$ as $f_{i,a}$

$\hat{T} = 1$
for $i = 1, 2 \dots T$ **do**
  for *each* $a \in A_i$ **do**
    if X_a, b_a *and* rc_a *are undefined* **then**
      Initialize X_a as identity matrix, b_a as zero vector and rc_a as 0
    end
    $S_a \leftarrow X_a^{-1} b_a$
    $D_{i,a} \leftarrow \frac{S_a^\top f_{i,a}}{1+\beta\left(\frac{rc_a}{\hat{T}}\right)^2} + \alpha\sqrt{f_{i,a}^\top X_{a-1} f_{i,a}}$
  end
  $a_{max} \leftarrow$ arm from A_i with the highest $D_{i,a}$ (selected arm)
  if $a_{max} == d_i$ **then**
    $\hat{T} \leftarrow \hat{T} + 1$
    $rc_{a_{max}} \leftarrow rc_{a_{max}} + 1$
    $X_{a_{max}} \leftarrow X_{a_{max}} + f_{i,a_{max}} f_{i,a_{max}}^\top$
    $b_{a_{max}} \leftarrow b_{a_{max}} + r_i f_{i,a_{max}}$
  end
end

rc_a indicates the number of times arm a has been recommended till round t and $\hat{T}$ is the number of recommendations till t rounds by the algorithm. The tuning parameter n is set to be 2 in this case. Thus $\frac{rc_a}{\hat{T}}$ gives a normalised form.

The cost for selecting arm a trade-off between click through rate and diversity in some sense. It discourages the optimal arm to be repeatedly selected and provides scope for the sub optimal ones to be recommended.

The DivLinUCB algorithm is constructed by introducing cost in LinUCB. The principal idea of standard UCB algorithms is to estimate an upper bound on the average reward for each arm. The upper bound is estimated using inequalities rooted in probability theory, such as Hoeffding's inequality.Similarly in this algorithm, expected payoff $D_{i,a}$ is calculated with $cost_{t,a}$ included. The arm with highest $D_{i,a}$ is taken as the optimal arm. This is done after maintaining rc_a and $\hat{T}$. In the subsequent rounds all these values are updated accordingly.

4 Experimental Results

4.1 Implementation

For implementing the LinUCB and DivLinUcB Algorithms for the dataset, we proceeded as follows:

- Parse each line of the input text file
- Since each line acts as an individual time step, we split the lines based on single space by which we obtained a list of 102 integers.
- Then we pop two values from the list one after another which gives us the arm chosen in that time step and reward corresponding to that arm respectively.
- Eventually, we are left with 100 elements left in the list which we assign as the context array in the corresponding time step.

To implement the algorithms we use the numpy and matplotlib libraries in python. Now, as we have obtained all the parameters required to perform the online prediction of the arms by calculating the coefficients, payout and standard deviation for each arm at every step and selecting the arm with the highest payout(i.e. Upper Confidence Bound) as the prediction. This prediction is followed by updating the matrix 'A' and the 1-D array 'b' corresponding to the predicted arm. We repeat this whole thing for all time steps.Here, we first we implemented the approach outlined in the Algorithm 1 followed by Algorithm 2.

4.2 Results

The explore-exploit mechanism is incorporated in our algorithm, by tuning the value of 'alpha' and 'beta', First we tried to identify the values of alpha which work the best for our algorithms, and the results for the following alpha values are displayed -

Accuracy of LinUCB for Different Values of Alpha

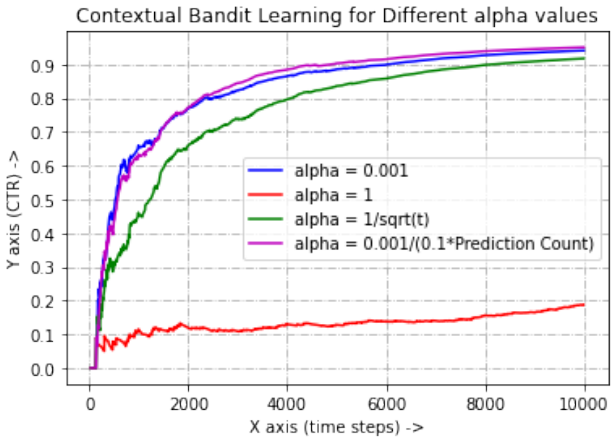

Figure 1

Accuracy of DivLinUCB for Different Values of Alpha

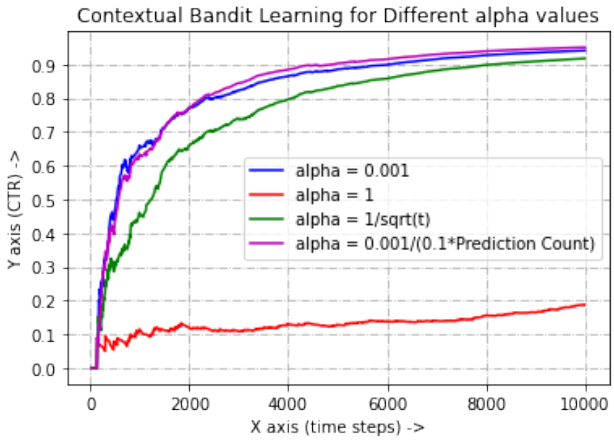

Figure 2

Now, we tried to identify the values of beta which work the best for our DivLinUCB algorithm, and the results for the following beta values are displayed -

Accuracy of DivLinUCB for Different values of beta

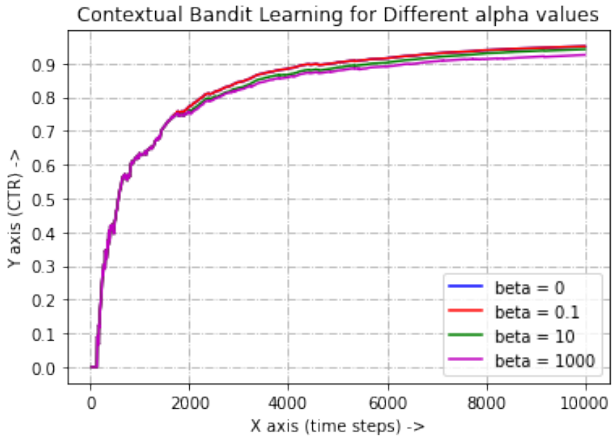

Figure 3

4.3 Observations

A comparative analysis of the various alpha values shows us how the accuracy of our algorithm varies based on different alpha values in Figure 1. As it is evident from the plots, the best CTR value was achieved with alpha as '0.001*(Number of Correct Predictions/10)' when the CTR value is 0.95 This is followed by the CTR values of 0.94, 0.91, and 0.20 for alpha values of 0.001, $1/\sqrt{t}$ and 1 respectively. The different alpha values were chosen to balance the explore-exploit trade-off.

Now when we change our alpha value as a function of the square root of time steps, we can see a significant improvement in our results. This is primarily because we are regulating our exploration as the time

passes and are limiting our predictions to exploit the most out of our trained algorithm with the passing time. When we use the alpha value of 0.001, we get even better results. The reason is that we are limiting the exploration to a very small value in this case, and exploiting the most.

To improve the CTR even further and achieve even better results on this dataset, we experimented by selecting the alpha value of 0.001, which has given us the best results till then, and tweaked it even further by dividing it by the count of the correct prediction for each arm. What this does is that it increases the exploitation, particularly for the arms which are giving us better results, and increased the exploration of the arms which have not been the best predictions, so far. This approach raised the CTR value of the LinUCB algorithm even further to 0.9508, which has been the best CTR rate of all the alpha values we've experimented with.

Reducing the alpha value even further from 0.001 wasn't very helpful, as it kept the CTR rate approximately the same. Which made me believe that the number of 0.001 was the sweet spot.

Another interesting pattern that can be observed in these plots is that the arm for which the UCB value is maximum has the most number of predictions, followed by the arm with the second-largest mean UCB and so on. This validates the fact that LinUCB selects the arm with the highest Upper Confidence Bound as the prediction. And, this is an indication of the fact that the implemented algorithm is correct.

We proceeded the same way in case of DivLinUCB. We observed that, for the tweaked alpha, we were able to raise the best CTR value. But, in case of tuning the model using different beta vaules and keeping alpha value fixed to the tweaked one, we observe very slight improvement in accuracy. But, as the beta value decreases towards 0, we see that the accuracy increases. Which expresses the fact that, the cost of selecting an arm at any time interval does not help us to improve that much as compared to the zero cost case in LinUCB.

5 Conclusion

From the experiments we did, it can be stated that, in case of both the algorithms i.e. LinUCB as well as the DivLinUCB algorithm, for a proper choice of alpha value i.e alpha=$\frac{0.001}{(correct\_pred[arm]+1)/10}$, we can improve the model performance drastically. In case of DivLinUCB, beta represented the effect of cost of arm selection. After experimenting with different beta values, we concluded that in this the accuracy of the model decreases as the beta value decreases to 0. Hence, in our dataset the introduction of arm selection cost was not effective.

END OF REPORT

6 References

- **Alexander Semenov, Maciej Rysz, Gaurav Pandey, Guanglin Xu. Diversity in news recommendations using contextual bandits**
- **Alekh Agarwal, Daniel Hsu ,Satyen Kale, John Langford, Lihong Li,Robert E. Schapire, Taming the Monster: A Fast and Simple Algorithm for Contextual Bandit**
- **Miroslav Dudik, Daniel Hsu ,Satyen Kale, Nikos Karampatziakis, John Langford,Lev Reyzin,Tong Zhang, Efficient Optimal Learning for Contextual Bandits**
- **Kevin A.,An Introduction to Contextual Bandits**
- **Contextual Bandit**
- **Additional Notes:**[www.cs.columbia.edu/~jebara/6998/Notes6.pdf
- **Dataset source:** [www.cs.columbia.edu/~jebara/6998/dataset.txt]