Collaborative Filtering with Contextual Bandits

Assignment 1 Report — Paris Dauphine PSL University

Zambra team:
Chen Dang
Công Minh Dinh
Oskar Rynkiewicz

Overview: Based on the paper *A Contextual-Bandit Approach to Personalized News Article Recommendation*[1], a contextual bandit algorithm LinUCB was implemented and analyzed in a recommendation system setting.

1 Introduction

Internet businesses can improve their quality of service by adapting their products to individual users. An optimized recommendation system incorporating content and user information is the key to provide the best service to their users.

Traditionally, recommendation systems are based on collaborative filtering, content-based filtering, or hybrid systems. A recent approach to optimization in personalized recommendation systems incorporate the Contextual Bandits algorithms. These algorithms can leverage the dynamic contextual information about items, users, and other data, to produce high-quality recommendations.

In our project, we focus on stochastic linear bandits with context. We study, implement, and analyze the LinUCB algorithm in two versions: with disjoint linear models, and with hybrid linear models. Both methods were introduced in the paper *A Contextual-Bandit Approach to Personalized News Article Recommendation*[1]. We will also apply the algorithms to a movie recommendation problem with *MovieLens* dataset [2] and evaluate the algorithms in terms of regret they produce.

2 Classic multi-armed bandit

The multi-armed bandit technique is an on-line algorithm that sequentially selects actions and learns which ones are more rewarding that others. An optimal policy seeks for a balance between exploiting known, good actions and unknown, potentially better actions. In the beginning of our work, we focused on context-free bandit algorithms.

2.1 Notation for stochastic context-free bandits

Stochastic bandit with K arms is a collection of K unknown distributions $(P_i)_{i=1}^{k}$. The i-th bandit's arm is associated with the i-th distribution P_i. We wish to learn which arms can generate the highest reward. Learning progresses throughout $t = 1, \ldots, T$ trials. After T trials, K collections of T independent, identically distributed random variables $X_{i,t} \sim P_i$ are produced.

Intially, we assumed $X_{i,t} = \mu_i + \varepsilon_t$ where μ_i is its mean value and $\varepsilon_t \sim \mathcal{N}(0, \sigma_2)$. Realization of $X_{i,t}$ is called a reward $r_{i,t}$. Set of actions $\mathcal{A} = \{1, \ldots, K\}$ has possible arms to pull. The strategy for choosing these actions is a policy denoted I_t. At each trial t, $a_t \in \mathcal{A}$ is choseen according to the distribution I_t.

Let X_{I_t} be a distribution of an arm choosen according to I_t. Having the policy dictate $I_t = i$, the result of drawing from $X_{I_t=i,t}$ is a received reward $r_{i,t}$. Different policies result in different expected reward $\mathbb{E}\left[r_{i,t} \sim X_{It}\right]$. The goal is maximize the expected total reward. Equivalently, we want to **minimize the expected total regret** $R(T)$:

$$R(T) = \mathbb{E}_{I_t}\left[\sum_{t=1}^{T} r_{i^*,t}\right] - \mathbb{E}_{I_t}\left[\sum_{t=1}^{T} r_{i,t}\right] \tag{1}$$

where $r_{i^*,t}$ is the optimal reward.

We used three strategies I_t:

- Random policy: $I_t^{\mathcal{U}} \sim \mathcal{U}\{1, K\}$
- ε-greedy policy:

$$I_t^{\varepsilon} = \begin{cases} \operatorname{argmax}_i \hat{\mu}_{i,t} & \text{w/prob} \quad 1-\varepsilon \\ I_t^{\mathcal{U}} \quad \text{w/prob} & \varepsilon \end{cases}$$

Where $\hat{\mu}_{i,t}$ is the mean estimate of arm i at time t and $\varepsilon = \frac{\alpha}{t}$ or $\varepsilon = \frac{\alpha}{\sqrt{t}}$, α is a constant.

- Upper Confidence Bound (UCB): Policy favors exploring arms that potentially can yield high reward. We always select the greediest action to maximize the upper confidence bound:

  $$I_t = \underset{i}{\operatorname{argmax}}\, \hat{\mu}_{i,t} + \hat{U}_{i,t}$$

  Where $\hat{U}_{i,t}$ the estimated upper bound. By Hoeffding's Inequality, we can state UCB1 algorithm as follows:

  $$I_t = \underset{i}{\operatorname{argmax}}\, \hat{\mu}_{i,t} + \sqrt{\frac{2\log t}{N_{i,t}}}$$

  where $N_{i,t}$ is the number of times the arm i was pulled until t.

2.2 Experimentation

The very first steps of our project were to build several simple algorithms for classic multi-armed bandit problem: random, ε-greedy and Upper Confidence Bound.

We assume that distributions $X_{i,t}$ are bounded $[0,1]$. The results obtained by implementation the algorithm is shown on Figure 2.2.

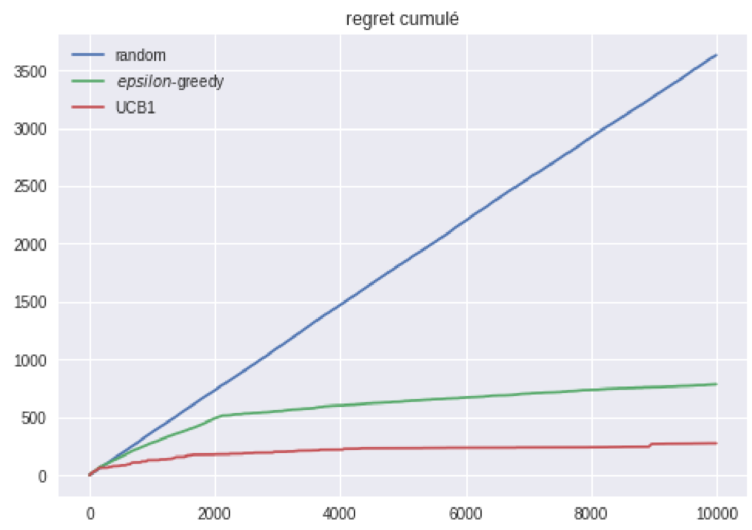


Figure 1: Comparison of three approaches for classic bandit problem

The results show that in the case of UCB, the regret is well minimized. We noted that UCB uses a smarter way to balance exploration and exploitation than random or ε-greedy strategy. Specifically, in each trial t, it estimates both the mean payoff $\mu_{t,a}$ of each arm a as well as a corresponding confidence interval $c_{t,a}$, so that:

$$|\mu_{t,a} - \mu_a| < c_{t,a}$$

holds with high probability. They then select the arm that achieves a highest upper confidence bound: $a_t = \text{argmax}_a(\mu_{t,a} + c_{t,a})$. It can be shown that UCB has a small total T- trial regret that is only logarithmic in the total number of trials T.

The algorithms presented in the next sections and applied to our recommendation system will then be built from the idea of UCB.

3 Contextual linear bandits

Real-world problems have *contextual data* such as information about user, actions, content. Additional information can improve the quality of the actions and lead to lower regrets.

The context observed by a learning algorithm, $\mathcal{C}$, is usually large, but has an underlying structure which can be exploited. In particular, it's effective to assume a linear structure.

The context could relate to user or actions, but in our approach we assume content-based filtering and use only the content features - actions. In previous formulation, an action was identified with its index. Now, we'll identify each action with vector $\mathbb{R}^d$ of d features. Our context, actions feature vectors, is static, although in general it doesn't need to be. The user is assumed unknown and we won't incorporate user context into the feature vector.

Movie recommendation setting

In this project, the arm are movies to serve users based on contextual information movies. As the learning progresses, the movie selection strategy is adapted in each iteration.

3.1 LinUCB with Disjoint Linear Models

We introduce the expected payoff of an arm a is linear in its d-dimensional feature $x_{t,a}$ with some unknown coefficient vector θ_a^*, namely for all t trial:

$$E[r_{t,a}|x_{t,a}] = x_{t,a}^T \theta_a^*$$

where:

- $a_t \in \mathcal{A}$ is defined as an arm or action, ie item to be recommended. In our case a presents movies.
- $r_{t,a}$ is reward of action a in trial t.
- $x_{t,a}$ is features describing both user and the selected arm a at trial t. It is referred to as the context.
- θ_a^* is an unknown coefficient vector specific to arm a.

Let (D_a, c_a) be training data, the goal of this model is to decrease the error between real reward and estimated reward. So θ_a is chosen in the following way:

$$\hat{\theta}_a = \underset{\theta_a \in \mathbb{R}^d}{argmin} (c_a - < \theta_a, D_a >)^2 + \lambda ||\theta_a||_2^2$$

where λ is the coefficient of ridge regression. The $\hat{\theta}$ is as follows:

$$\hat{\theta}_a = (D_a^T D_a + \lambda \times I_d)^{-1} D_a^T c_a$$

This model is called disjoint since the parameters are not shared among different arms. We calculate the coefficient, payout and standard deviation for each arm at every step and select the arm with the highest payout (i.e. Upper Confidence Bound) as our prediction. A new observation (x_t, a_t, r_t) is improved with each repeated time step. In our implementation, the prediction is given by an update of matrices A and b for the predicted arm.

Algorithm 1 LinUCB with disjoint linear models.

```
0:  Inputs: $\alpha \in \mathbb{R}_+$
1:  for $t = 1, 2, 3, \ldots, T$ do
2:      Observe features of all arms $a \in \mathcal{A}_t$: $\mathbf{x}_{t,a} \in \mathbb{R}^d$
3:      for all $a \in \mathcal{A}_t$ do
4:          if $a$ is new then
5:              $\mathbf{A}_a \leftarrow \mathbf{I}_d$ ($d$-dimensional identity matrix)
6:              $\mathbf{b}_a \leftarrow \mathbf{0}_{d\times 1}$ ($d$-dimensional zero vector)
7:          end if
8:          $\hat{\boldsymbol{\theta}}_a \leftarrow \mathbf{A}_a^{-1}\mathbf{b}_a$
9:          $p_{t,a} \leftarrow \hat{\boldsymbol{\theta}}_a^\top \mathbf{x}_{t,a} + \alpha\sqrt{\mathbf{x}_{t,a}^\top \mathbf{A}_a^{-1}\mathbf{x}_{t,a}}$
10:     end for
11:     Choose arm $a_t = \arg\max_{a\in\mathcal{A}_t} p_{t,a}$ with ties broken arbi-
        trarily, and observe a real-valued payoff $r_t$
12:     $\mathbf{A}_{a_t} \leftarrow \mathbf{A}_{a_t} + \mathbf{x}_{t,a_t}\mathbf{x}_{t,a_t}^\top$
13:     $\mathbf{b}_{a_t} \leftarrow \mathbf{b}_{a_t} + r_t\mathbf{x}_{t,a_t}$
14: end for
```

We also noted that in LinUCB disjoint, the parameters are not shared between different arms; and the solution is reached after applying ridge regression. It is also proved that if the arm set $\mathcal{A}_t$ is fixed and contains K arms, then the confidence interval decreases fast and prove the strong regret bound of $\tilde{O}(\sqrt{KdT})$, where d is the dimension of features and T the time. In the next section, a more complex model is studied with the ability of sharing parameters among different arms.

3.2 LinUCB with Hybrid Linear Models

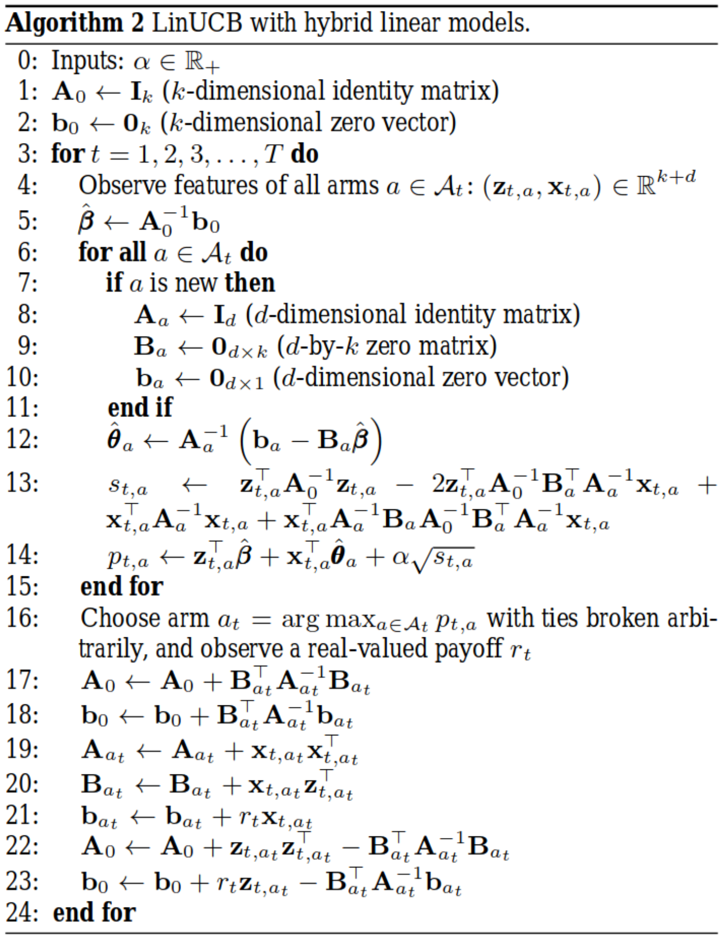
In many applications, it is helpful to use features that are shared by all arms, in addition to the arm-specific ones (as in the previous section). Formally, we adopt the following hybrid model by adding another linear term:

$$E[r_{t,a}|x_{t,a}] = z_{t,a}^T\beta^* + x_{t,a}^T\theta_a^*$$

where:

- $z_{t,a}$ is features of the current user and movies combination.
- β^* is an unknown coefficient vector common to all arms.

To implement LinUCB Hybridge, compared to LinUCB disjoint, we introduce here the new B matrix shared by all features.



Algorithm 2 LinUCB with hybrid linear models.

```
 0: Inputs: $\alpha \in \mathbb{R}_+$
 1: $\mathbf{A}_0 \leftarrow \mathbf{I}_k$ ($k$-dimensional identity matrix)
 2: $\mathbf{b}_0 \leftarrow \mathbf{0}_k$ ($k$-dimensional zero vector)
 3: for $t = 1, 2, 3, \ldots, T$ do
 4:   Observe features of all arms $a \in \mathcal{A}_t$: $(\mathbf{z}_{t,a}, \mathbf{x}_{t,a}) \in \mathbb{R}^{k+d}$
 5:   $\hat{\boldsymbol{\beta}} \leftarrow \mathbf{A}_0^{-1}\mathbf{b}_0$
 6:   for all $a \in \mathcal{A}_t$ do
 7:     if $a$ is new then
 8:       $\mathbf{A}_a \leftarrow \mathbf{I}_d$ ($d$-dimensional identity matrix)
 9:       $\mathbf{B}_a \leftarrow \mathbf{0}_{d\times k}$ ($d$-by-$k$ zero matrix)
10:       $\mathbf{b}_a \leftarrow \mathbf{0}_{d\times 1}$ ($d$-dimensional zero vector)
11:     end if
12:     $\hat{\boldsymbol{\theta}}_a \leftarrow \mathbf{A}_a^{-1}\left(\mathbf{b}_a - \mathbf{B}_a\hat{\boldsymbol{\beta}}\right)$
13:     $s_{t,a} \leftarrow \mathbf{z}_{t,a}^\top\mathbf{A}_0^{-1}\mathbf{z}_{t,a} - 2\mathbf{z}_{t,a}^\top\mathbf{A}_0^{-1}\mathbf{B}_a^\top\mathbf{A}_a^{-1}\mathbf{x}_{t,a} +$
        $\mathbf{x}_{t,a}^\top\mathbf{A}_a^{-1}\mathbf{x}_{t,a} + \mathbf{x}_{t,a}^\top\mathbf{A}_a^{-1}\mathbf{B}_a\mathbf{A}_0^{-1}\mathbf{B}_a^\top\mathbf{A}_a^{-1}\mathbf{x}_{t,a}$
14:     $p_{t,a} \leftarrow \mathbf{z}_{t,a}^\top\hat{\boldsymbol{\beta}} + \mathbf{x}_{t,a}^\top\hat{\boldsymbol{\theta}}_a + \alpha\sqrt{s_{t,a}}$
15:   end for
16:   Choose arm $a_t = \arg\max_{a\in\mathcal{A}_t} p_{t,a}$ with ties broken arbi-
      trarily, and observe a real-valued payoff $r_t$
17:   $\mathbf{A}_0 \leftarrow \mathbf{A}_0 + \mathbf{B}_{a_t}^\top\mathbf{A}_{a_t}^{-1}\mathbf{B}_{a_t}$
18:   $\mathbf{b}_0 \leftarrow \mathbf{b}_0 + \mathbf{B}_{a_t}^\top\mathbf{A}_{a_t}^{-1}\mathbf{b}_{a_t}$
19:   $\mathbf{A}_{a_t} \leftarrow \mathbf{A}_{a_t} + \mathbf{x}_{t,a_t}\mathbf{x}_{t,a_t}^\top$
20:   $\mathbf{B}_{a_t} \leftarrow \mathbf{B}_{a_t} + \mathbf{x}_{t,a_t}\mathbf{z}_{t,a_t}^\top$
21:   $\mathbf{b}_{a_t} \leftarrow \mathbf{b}_{a_t} + r_t\mathbf{x}_{t,a_t}$
22:   $\mathbf{A}_0 \leftarrow \mathbf{A}_0 + \mathbf{z}_{t,a_t}\mathbf{z}_{t,a_t}^\top - \mathbf{B}_{a_t}^\top\mathbf{A}_{a_t}^{-1}\mathbf{B}_{a_t}$
23:   $\mathbf{b}_0 \leftarrow \mathbf{b}_0 + r_t\mathbf{z}_{t,a_t} - \mathbf{B}_{a_t}^\top\mathbf{A}_{a_t}^{-1}\mathbf{b}_{a_t}$
24: end for
```

4 Implementation & Experiments

4.1 Data processing

In our project, we will use the Movielens dataset, collected by GroupLens Research. Because of the time required to execute our algorithm, we will use the classification of the 1M movies dataset. It has 1 million votes of 6000 users on 4000 films and was published in February 2003.

Due to the computational requirements, we firstly take a small part of the dataset and we also note some pre-treatments for this set:

- Ratings are resized between 0 and 1
- The movies watched by more than 1000 users were selected
- Users who watched more than 150 movies were selected
- The 30 most important features have been chosen

4.2 User & Films modeling

First of all, the singular value decomposition (SVD) is applied on the ratings matrix in order to get user and movie features.

Suppose we have M the rating matrix, set W the diagonal matrix then we can decompose M:

$$M = \phi_u^T W \phi_a$$

where ϕ_u and ϕ_a are the corresponding feature vectors of users and films.

In our study, we suppose that the features of films are known factors and user information is unknown. So the 30 most important features of movies are used in our study. As we have already selected the most viewed movies and most active users, the sparsity of the rating matrix is about 81%. For the movies occasionally not rated by certain user, we return the average rating of all other users who have rated. This is easy to implement but not a very accurate estimation, while considering that this case didn't happen very often, we still adopted this approach.

4.3 Results & Evaluation

4.3.1 LinUCB with disjoint linear models

Firstly, we tested Movielens dataset with disjoint model. With one arbitrary user, the accumulated regret is shown as Figure 2. We noticed that with more iterations, there will be fewer regrets. The most frequently recommended films and their rankings are shown in Figure 3. Among these films, these two most recommended are the highest rated. Figure 4 shows the recommended percentage of movies. We noticed that over 80% are rated 1.0 which corresponds to the highest score, the rest being 0.8. According to Figure 5, the average rating of movies recommended are higher and more likely to be 1.0 as the number of iterations increases.

Figure 2: Accumulated Regret of LinUCB disjoint linear model

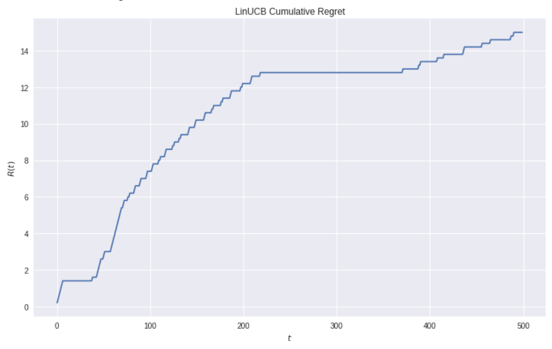


Figure 3: Movies most frequently recommended

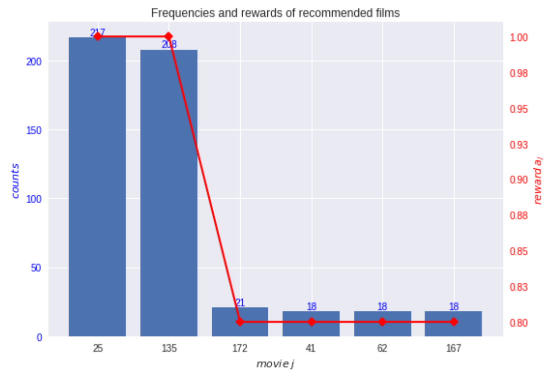


Figure 4: Ratings of Movies most frequently recommended

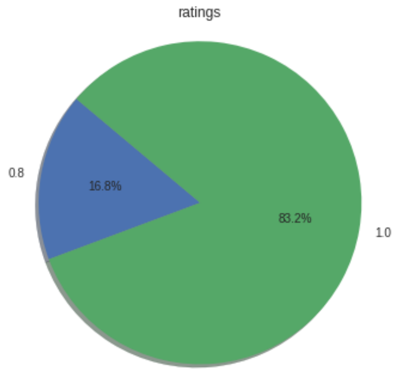


Figure 5: Ratings of LinUCB disjoint linear model

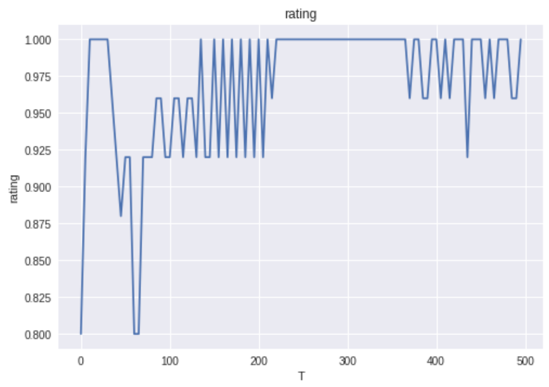
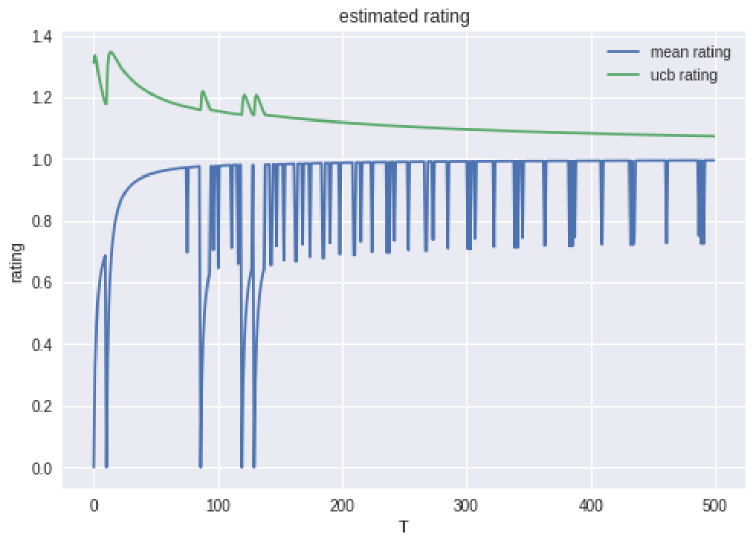


Figure 6 shows the evolution of estimated ratings of recommended movies during each iteration. As more iterations we have, the mean rating is more closer to 1.0 which is its real rating, and upper confidence bound diminishes, which corresponds to the hypothesis that more evaluations we have, more precisely we know about the true rating of the movie.

Figure 6: Estimated Ratings of movies recommended in LinUCB disjoint linear model



Parameters test

We also tested the influence of different α, which is the coefficient of upper confidence bound, and the noise. From Figure 7 we found that when $\alpha = 1.4$ we have lowest accumulated regret. But it should be noted that the performance is not very stable and because the low exploration rate makes it stick to the best choice it has instead of exploring other movies. The second best is when $\alpha = 1.8$, the algorithm has balanced exploitation and exploration, and achieved satisfying result.

Figure 8 shows that bigger the noise, more accumulated regret it will be. When $\delta = 0.01$, the influence was almost negligible. But when $\delta = 0.1$, it took much more unstable.

Figure 7: linUCB disjoint alpha comparison

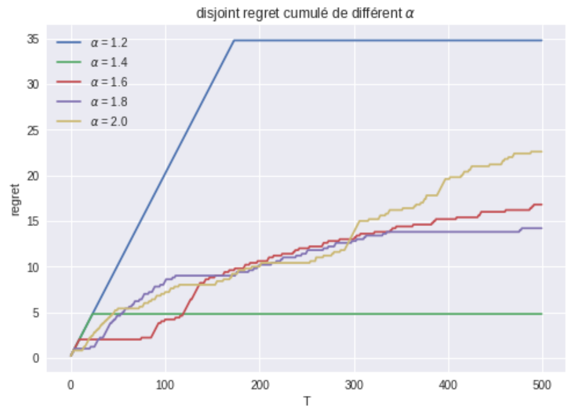
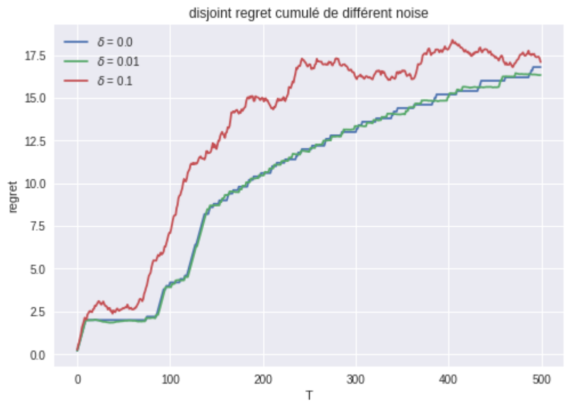


Figure 8: linUCB disjoint noise comparison



4.3.2 LinUCB with hybrid models

With LinUCB with hybrid models, we achieved better result. Figure 9 shows the evolution of ratings of movies recommended. Comparing to the previous figure of disjoint models, this result shows much better performance and more stable recommendation. the ratings are always greater than 0.8 and when continues to have great recommendation later even when it explores.

Figure 9: Ratings of LinUCB hybrid linear model

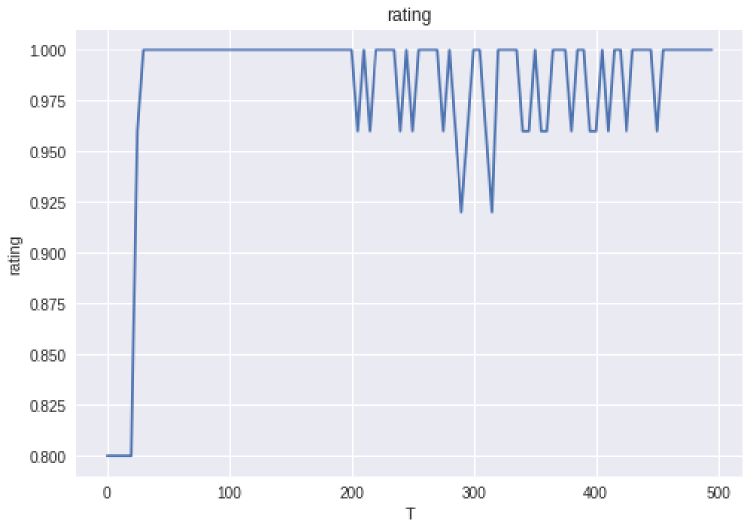


Figure 10 shows the estimation ratings of films recommended. Comparing to the previous model, we now converges much faster. Figure 11 show the percentage of ratings of movies recommended. It has higher percentage of 1.0 rated movies than the previous model.

Figure 10: Estimated Ratings of movies recommended in LinUCB hybrid linear model

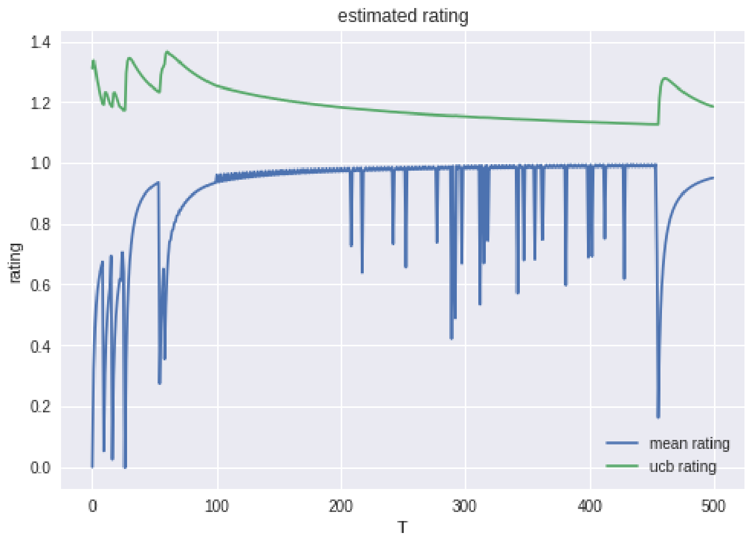
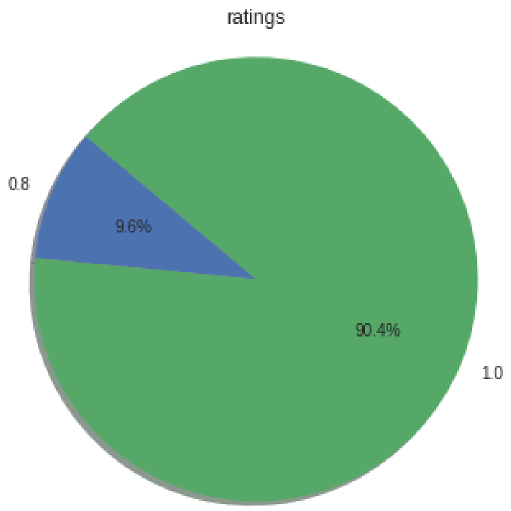


Figure 11: Ratings of Movies most frequently recommended



For each alpha value, we plotted the Regret and compare the different alpha values to show how the accuracy of our algorithm varies.

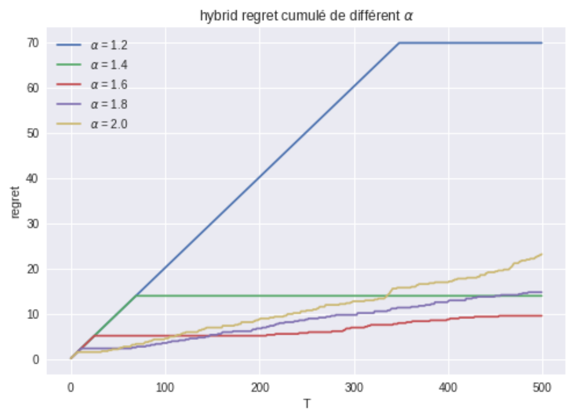


Figure 12: linUCB hybrid alpha comparison

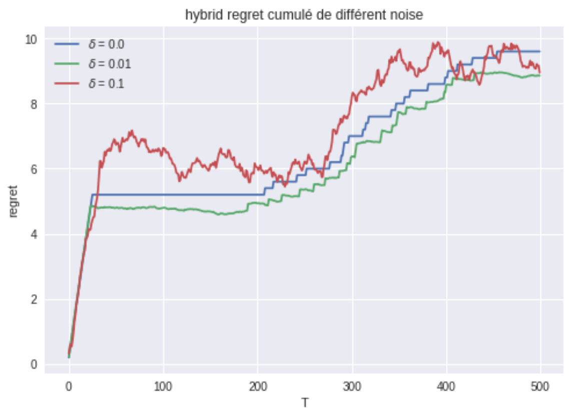


Figure 13: linUCB hybrid noise comparison

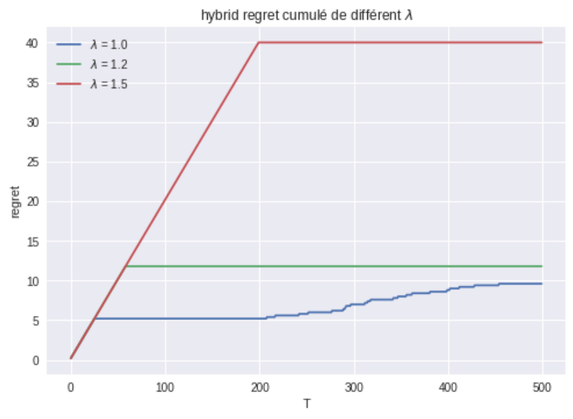
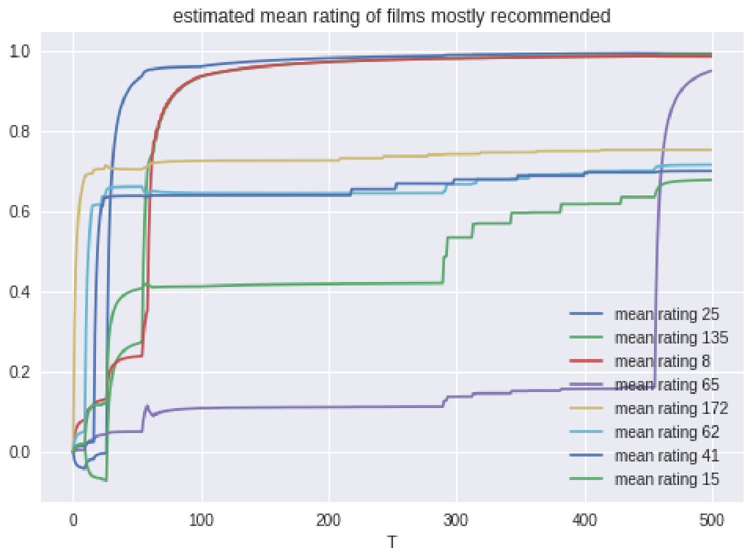


Figure 14: linUCB hybrid lambda comparison

We found when λ and θ are low, the result obtained is better. $\alpha = 1.6$ gives overall best accumulated regret. Smaller α is, more it is likely to choose exploitation instead of exploration. As a result, with smaller α, the algorithm usually satisfies with the current reward, even if there could be other arms which have much bigger reward. In contrast, if α is too high, it will always choose to exploration whether current choice is good or not, therefore causes more regret.

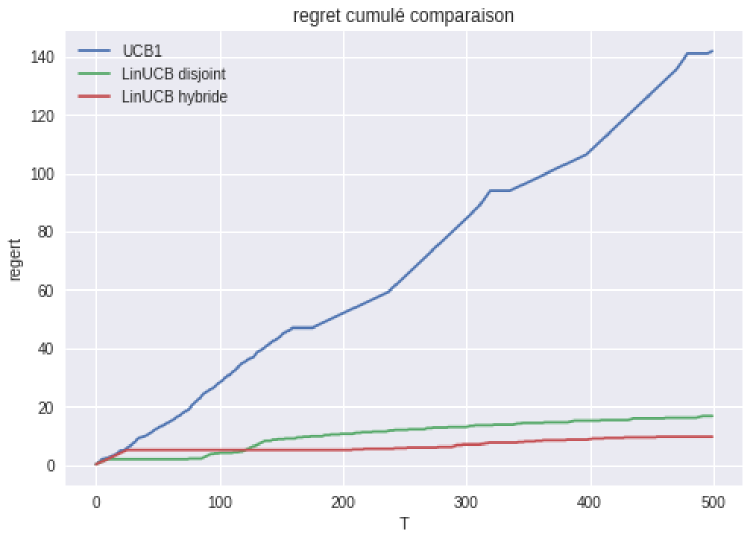
Figure 15 shows the evolution of estimated ratings of some movies recommended. It shows the overall tendency of how each film is rated during the algorithm.

Figure 15: linUCB hybrid evolution of estimated ratings of movies mostly recommended



4.3.3 Comparison between approaches

Figure 16: regret comparison of different approaches



We see that LinUCB is clearly better than UCB1. With more iterations, LinUCB Hybrid became better than LinUCB disjoint.

5 Conclusion

The project concerns recommendation systems, which are highly relevant in today's world. We explored a stochastic contextual bandit approach for creating such systems. The UCB1 and LinUCB algorithms with disjoint and hybrid models were implemented in Python. Amongst these solutions, the hybrid version of LinUCB gave the best results, followed closely by the disjoint LinUCB. Both LinUCB algorithms exceeded greatly the UCB1 algorithm, proving that the movies contextual information can improve the quality of recommendations. The choice of parameters is important, as we noted that they can drastically impact the results. Contextual linear bandits are a rich family of algorithms, which has a lot to offer. In particular, we note that the choice of movie features as the only contextual information might be limiting. In future work, expanding the context with other information would be worthwhile.

References

[1] Li, Lihong, et al. "A contextual-bandit approach to personalized news article recommendation." Proceedings of the 19th international conference on World wide web. ACM, 2010. https://arxiv.org/abs/1003.0146

[2] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets https://grouplens.org/datasets/movielens/1m/