Linear Bandit for Recommendation Systems

Data Science Lab

Supervisors : Benjamin Negrevergne - Alexandre Vérine

Adrien Golebiewski – Linda Gutsche – Sihan Xie

Abstract

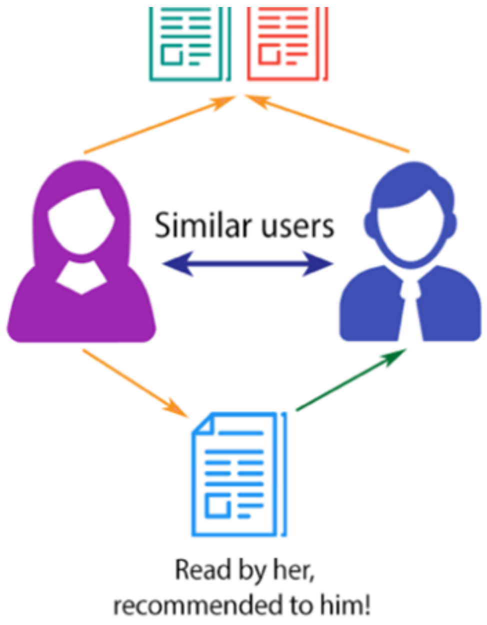
A large number of online services provide automated recommendations to help users navigate through a large collection of items. New items (products, videos, songs, advertisements) are suggested on the basis of the user's browsing history. Recommendations have to satisfy the dual goal of helping the user explore the space of available items, while allowing the system to probe for the user's preferences.

Here is our report for the assignment on recommender-systems. We chose the Linear Bandits approach and studied the trade-off between exploration and exploitation through various linearly and non linearly parameterized multi-armed bandits algorithms.

Dauphine | PSL
UNIVERSITÉ PARIS

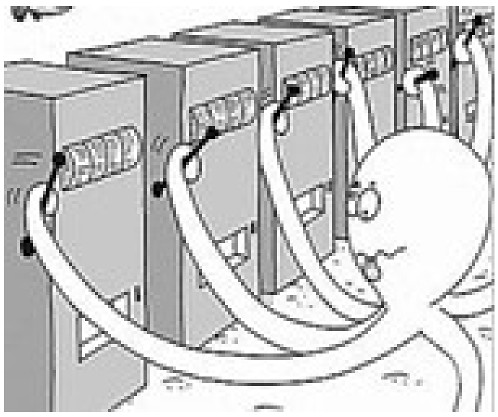
For October 16th, 2022

1 Introduction



Anyone who has shopped on Amazon or browsed for a binge worthy TV series on Netflix has experienced personalized recommendations. Companies use recommender-systems to increase customer engagement and revenue. These algorithms take user, product, and contextual metadata as input and produce personalized, dynamic content as output through collaborative or content-based filtering methods. After gathering data on a user preferences, a collaborative filtering strategy can make recommendations to a viewer based on content that has been liked by other viewers with similar tastes. In each case, the need for collecting viewer-specific data diminishes the effectiveness of these systems until such data can be gathered. This is typically referred to as the "cold start problem."

Multi-armed bandit algorithms is one of the common framework to approach the trade-off between gathering data and exploiting it: A gambler enters a casino with a fixed amount of money and wants to learn the best strategy to maximize their profits. They are presented with a row of k slot machines, with each machine having a hidden payoff function that determines how much it will pay out. The gambler iteratively plays one lever per round and observes the associated reward. The objective is to maximize the sum of the collected rewards.



Initially the gambler has no information about which machine is expected to pay out the most money, so they try one at random and observe its payout. Now that they have a little more information than before, they need to decide: do they exploit this machine now that they know more about its payoff function, or do they explore the other options by pulling arms that they have less information about?

The Multi-armed bandit (or MAB) problem models an agent that simultaneously attempts to acquire new knowledge (exploration) and optimize their decisions based on existing knowledge (exploitation). The agent balances these competing tasks in order to maximize his total value over the period of time considered. How the trade-off between exploration and exploitation is achieved depends on the particular bandit algorithm. Usually, MAB problems are analyzed on the basis of **regret**, that is the difference of the total reward achieved by always selecting the optimal arm and the total reward achieved by the algorithm.

In this report, we focus on applications of MAB algorithms in recommendation systems. We study the balance between exploitation and exploration by developing some algorithms of Bandits framework applied in a real context of movie recommendations through the MoviesLens dataset.

All algorithms were coded from scratch on Python, using only the help of the libraries `random`, `numpy`, `pandas` and `matplotlib`.

2 Our framework and some quick presentation of the Bandit algorithms

2.1 Framework

We will place ourselves in a linear framework: we will assume that the expected payoff of an arm is linear in its d-dimensional feature vector X with some unknown coefficient vector θ.

For the epsilon-greedy and upper confidence bound (UCB) algorithm, we chose to use a context free model. For Linear UCB, for both the independent arms version and for the hybrid one, we considered the context and used the movie features provided by the database.

All algorithms work through trial and error: at each trial t, the algorithm picks a movie to recommend to the user, and used the feedback it gets to estimate the average payoff $r_{t,a}$ of each arm a.

2.1.1 epsilon-greedy

In each trial, the ϵ-greedy algorithm chooses the greedy arm (i.e., the arm with highest payoff estimate) with probability $1 - \epsilon$, and with probability ϵ, it chooses a random arm.

2.1.2 UCB

Upper confidence bound algorithms estimate not only the mean payoff $r_{t,a}$, but also a corresponding interval $c_{t,a}$, so that $|r_{t,a} - r_a| < c_{t,a}$ holds with a high probability. The algorithm then selects the arm that achieves the highest upper confidence bound $argmax_a r_{t,a} + c_{t,a}$.

To compute $c_{t,a}$ we chose the version of UCB that estimates $c_{t,a} = \alpha\sqrt{\frac{log(t)}{N_{t-1}(a)}}$ where $N_{t-1}(a)$ is the number of times the arm a has already been tried. α is a constant, the confidence parameter.

2.1.3 LinUCB, regular and hybrid - version with context

The precise algorithms can be found in [3]. With $x_{t,a}$ a feature vector that represents the information we have on the given user in respect to the arm a,

- The regular version tries to estimate $\mathbb{E}[r_{t,a}|x_{t,a}] = x_{t,a}^T\theta_a^*$ (linear model, with independent arms).
- The hybrid version tries to estimate $\mathbb{E}[r_{t,a}|x_{t,a}] = z_{t,a}^T\beta^* + x_{t,a}^T\theta_a^*$ (linear model, with crossing features)

3 Experiments & Results

In this section, we study the behavior and the performance of the context-free and contextual models presented in the previous section on the *MovieLens* dataset. We start with an introduction of the dataset and the problem setting. Then, we describe performance metrics and feature vector generation methods. Finally, we compare the results of the different algorithms by means of a graphical representation.

3.1 Descriptive Analysis of the *MovieLens* dataset

MovieLens is a rating dataset from the MovieLens website, which has been collected over some period. It contains 25 millions ratings from 162 000 users on 62000 movies. We loaded the data into dataframes using the library pandas. We analyzed and visualized it to draw some key insights to better understand the impact of the given data on our study of the recommendation algorithms.

We can observe that most of the users have rewarded movies they watched with a 4 star rating and followed by 3 star and 5 star. The same has been displayed below using a pie chart to understand the contributions. We have also seen that that very few movies were watched by more than 100 out of 943 users. The graph confirms our observations of a high volume of movies with a low rating count.

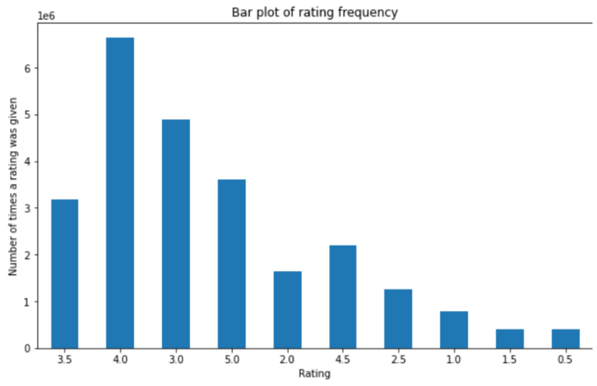


Figure 1: Bar plot of rating frequency

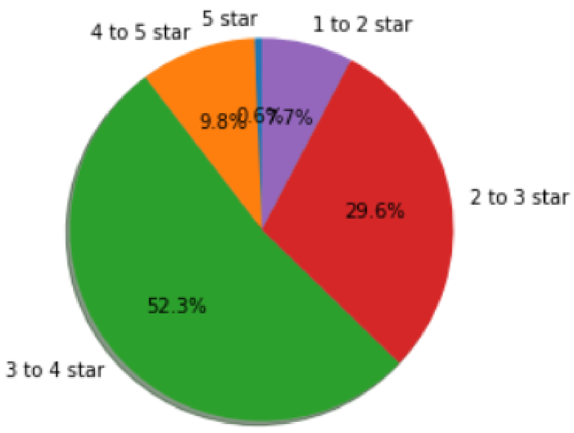


Figure 2: Split of movies count based on their overall average rating

These post-development results allow us to have a better vision of the dataset and have allowed us to define our development approach and to better adapt our algorithms.

3.2 Feature construction

We now define the method we used to construct the user/movie feature vectors for the disjoint and hybrid LinUCB models, which can be seen as the hardest part as the quality of these feature vectors will affect the learning speed of the model. Ideally, in the experiment of Yahoo, user features include search histories at an aggregated level as well as demographic information and movie feature includes category information [1]. However, since the *MovieLens 25M Dataset*[2] doesn't contain user information, our feature vectors could only be generated from user rating records and movie genre.

The first method is to apply the matrix factorization to ratings, since from an interactive aspect, each user can be interpreted as a feature of the movie, each movie can also be interpreted as a feature of the user. We retrieved user/movie features by using Non-negative matrix factorization (NMF) where a matrix V(rating) is factorized as the product of the matrices W(user features) and H(movie features). To determine the best k (latent features dimension), we used cross-validation to compute the Frobenius distance of $\|V - WH\|^2$as a measure of matrix approximation error.

Additionally, each film could also be represented by a raw feature vector consisting of approximately 20 dummy variables, constructed by one-hot encoding on movie genre. After that, since these features are highly correlated, we also tried Principal component analysis (PCA) to reduce the dimensionality of the generated features to see if the performance of the model could been improved.

3.3 Results of different models

3.3.1 Context-free model

We started with the stochastic context-free models to understand the exploration/exploitation dilemma of the bandit problem. As we can see from the figures, within 3000 steps, ϵ-greedy and UCB don't converge and they couldn't fully learn the dataset. We can also verify that with a higher hyperparameter ϵ or α, the model is more inclined to go exploring so it will have a higher regret in the early stages.

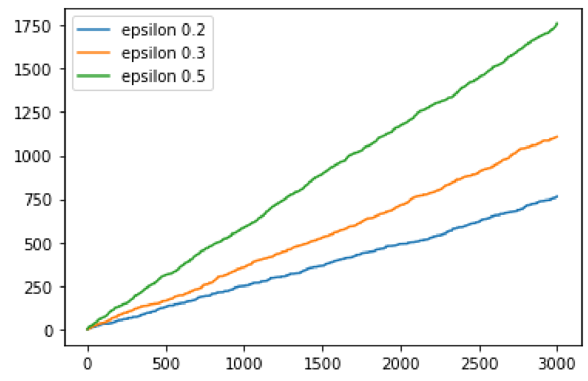


Figure 3: ϵ-greedy: cumulative regret for different ϵ

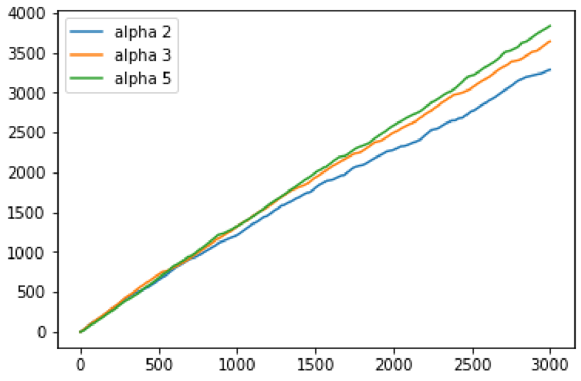


Figure 4: UCB: cumulative regret for different α

3.3.2 Contextual-Bandit model

LinUCB with Disjoint Linear Models

We first tested disjoint LinUCB with the different generated feature vectors introduced in 3.2. In figure 3, the orange curve represents the features generated by NMF, the blue represents the features generated by one-hot encoding of movie genre, and the green represents the features after dimensionality reduction. It seems that the features generated with NMF help the model to learn more quickly. As LinUCB is an extension of UCB, we can observe from figure 4 that it has the same property as UCB, the higher the α, the more it tends to explore. We can also conclude that after adding context to the model, LinUCB learns faster than the context-free model but at the cost of significant increase in execution time.

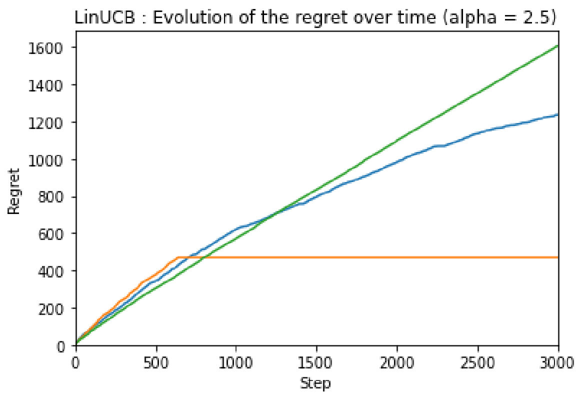


Figure 5: Disjoint LinUCB: cumulative regret for different generated features

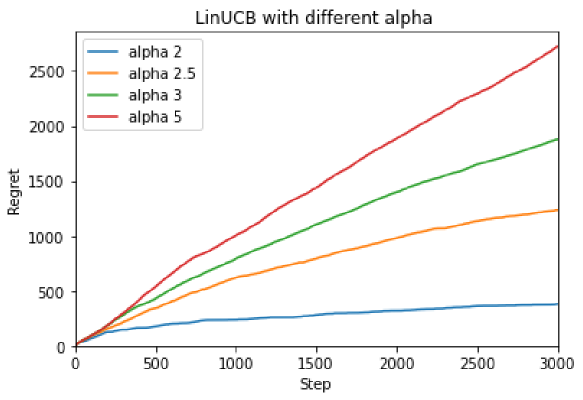


Figure 6: Disjoint LinUCB: cumulative regret for different α

LinUCB with Hybrid Linear Models

Hybrid LinUCB, compared to Disjoint LinUCB, may be more unstable in the early stage under the influence of shared parameter β, but it learns and converges faster.

In our previous attempts, we did not prohibit the model from recommending a movie multiple times to a user, as in the bandit problem, the model is able to have a more accurate estimate of the reward of each item through multiple trials. However, this is not realistic for recommender system, that usually recommend a movie to a user a limited number of times because, unlike the bandit algorithms, the goal is not to make sure that a user really likes a movie by recommending this movie a hundred times, since the user will not want to watch a hundred times the same movie. This is also why Hybrid LinUCB outperforms other models: at each time step the reward we receive for a movie provides information not only about itself, but also about all other movies, enabling us to forbid recommending twice the same movie as receiving the ratings of one helps estimating the ratings of others.

Figure 8 shows the situation if we prevent users from selecting the same movie. Let's interpret it : we observe that all the movies with a rating of 5 are recommended before timestep 600, no movie with a rating less than 1.5

is recommended before timestep 400. This achievement is beyond the reach of stochastic models. But when all the high-rated movies have been recommended, the model will eventually start recommending low-rated movies because it can only recommend the movies from the unpopular ones. So due to the limited dataset and our offline learning method, we can not go further, but we can still conclude that the hybrid LinUCB performs better.

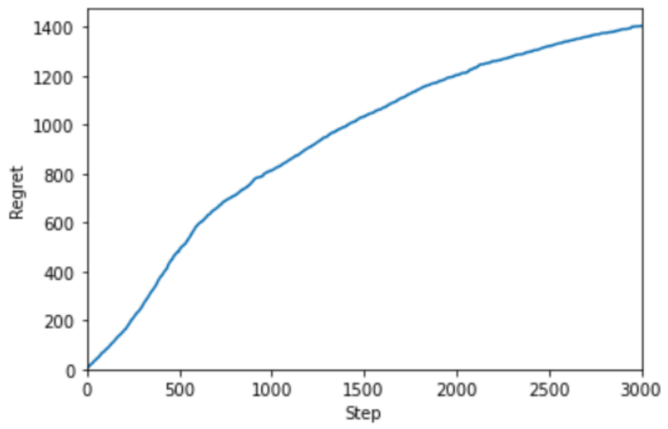


Figure 7: Hybrid LinUCB: cumulative regret

Evolution of the regret over time

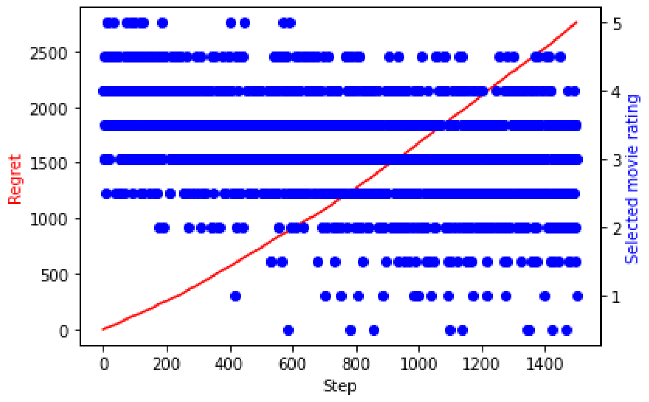


Figure 8: Hybrid LinUCB: cumulative regret

Evolution of the regret over time
Forbidding recommending more than once the same movie to the same user

3.4 Limitation of the study

- We limited ourselves to one user, to considerably simply our implementation, and as it seemed sufficient to highlight the characteristics of the different algorithms
- Due to the choice of the Bandit approach, and the limitation of using a dataset instead of live users, we had to restrict ourselves to recommending movies the user gave a grade to in the dataset

4 Project review

Given the time constraints, we were not able to explore as many approaches as we wanted, and due to limited computational power, we were not successful in comparing the linear model to non-linear model (ex : Neural Contextual Bandits Model) as we first planned to. We also regret to not have the time to create an implementation applicable to a real-world recommendation system (with several users, a time line, pertinent limitations on number of times a movie can be recommended, and a method of dealing with unreviewed movies that distinguishes between the user discarding the recommendation, watching only part of the movie, or watching it till the end but not reviewing it), as it finding a solution for that was a stimulating challenge.

The subject of Multi Armed Bandit was completely new to us, and it was very rewarding to read about it to get a good understanding of the topic, and to try and figure out ways to transcribe mathematical equations and results into working algorithms.

References

[1] Xiao Bai, B. Barla Cambazoglu, Francesco Gullo, Amin Mantrach, and Fabrizio Silvestri. Exploiting search history of users for news personalization. *Information Sciences*, 385-386:125–137, 2017.

[2] https://grouplens.org/datasets/movielens/25m/. Movielens 25m dataset.

[3] John Langford Lihong Li, Wei Chu and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. *ArXiv*, abs/1003.0146, 2010.