Project Report
Hierarchical Bandits for Music Recommendation

Yiming Fang, Su Su Hlaing
Email: {yf2484, sh3857}@columbia.edu

December 29, 2022

1 Introduction

Multi-armed bandits (MABs) are a framework for sequential decision-making and are often used for building recommendation systems. However, in applications such as music recommendation, it is often the case that the dataset contains thousands of items to choose from; if each item is modeled as an individual arm, the learning process would become extremely inefficient or even infeasible.

In this project, we propose a novel recommendation scheme called hierarchical bandits, and apply the technique to a real-world music dataset. Hierarchical bandits alleviate the problem of large sample spaces by partitioning the sample space in a hierarchical way and breaking up the decision-making process into a group of decision-making choices with different levels of granularity. By doing so, we can ensure that each sub-sample space contains a reasonable number of choices, such that well-known algorithms such as ϵ-decreasing can be applied effectively.

The rest of this report is organized as such: Section 2 formulates the data preprocessing and learning procedures in an abstract way. Section 3 discusses the implementation of the hierarchical bandits and the application in music recommendation. Section 4 discusses the strengths and weaknesses of this new approach and concludes the report.

2 Method Formulation

2.1 Data Preprocessing

In order to apply the method of hierarchical bandits, the dataset itself has to be organized in a hierarchical manner. This requires that we first preprocess the input dataset into a tree structure. In some cases, the dataset contains categorical organization that we can use directly. For example, in the case of music recommendation, the music genre and artist information can be used to group songs together. However, the native categories might lack the degree of granularity that we desire, and we need to further divide up the dataset. In such cases, we utilize KMeans as a method to explicitly segment data into a predefined number of clusters, and use the resulting clusters as an intermediate level in the data tree.

As shown in the example Figure 1, the data is segmented into 4 levels of clusters, where L0-cluster is the root, and L3-cluster is the leaves, or the raw input data. The intermediate levels represent some internal organization of the dataset, in terms of categories and sub-categories.

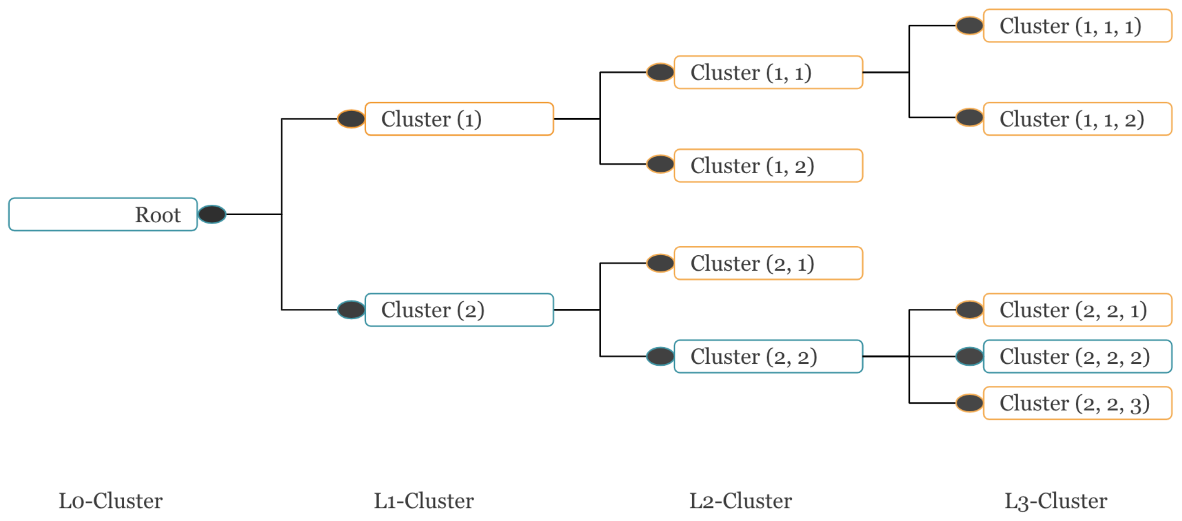


Figure 1: The data tree

2.2 Learning

Hierarchical bandits are a generalization of the traditional MAB methods. Whereas in traditional K-armed bandits, the agent chooses one action among K possible actions, each associated with an associated reward at each timestep, in the case of hierarchical bandits, the agent recursively chooses a series of actions until it reaches the leaf level. Only the leaf level is associated with the distribution of rewards. Each sub-branch encountered on the way will maintain its own set of Q values, which get updated each time the sub-branch is traversed.

Formally, assume that there are N possible choices at the leaf level (for example, assume that there are N possible songs to recommend), and the dataset is evenly partitioned into a k-way tree. Then the depth of the tree will be $\log_k N$, and each node will have k children. This implies that in order for the agent to make one action, it will make $log_k N$ decisions down the tree, and each decision involves making a choice among k possible actions. The new method allows us to reduce the sample space from N to k, which is a controllable quantity that we choose during the segmentation phase.

As in traditional MAB methods, hierarchical bandits work by gathering data using exploration and exploitation modes. The difference is that, instead of deciding whether to explore/exploit once, now we need to decide on each level. In other words, the probability of exploiting at level-i becomes $(1-\epsilon)^i$, because the probability of arriving at the current sub-branch with the exploiting mode is already $(1-\epsilon)^{i-1}$. Notice that the L0-cluster (root) is trivially chosen always. In Figure 1, we have shown an example path (green border) that is learned by the agent, and the rest of the branches are places for exploration (orange border).

3 Case Study: Music Recommendation

3.1 Exploratory Data Analysis

The dataset we used is [1], containing 2000 songs that are popular on Spotify. Besides basic information about the song's title and artist name, the dataset also contains information about its genre, as well as a number of quantitative sound attributes, including beats per minute, danceability, energy, acousticness, valence, etc.

Recall that we want to partition the input dataset into a tree structure. In our case, the obvious choice is to have genre and artist each serve as an intermediate level in the tree, with genre being a superset of artist. The rationale behind this choice is that the genre information is useful in grouping together similar songs, and the same artist is very likely to make songs that belong to one or few genres. The music listener who enjoys certain songs will be more likely to find songs in the same genre/made by the same artist enjoyable.

The ratio between artists and genres (731) is appropriate because $k \approx 731/149 \approx 5$. The ratio between songs and artists is also appropriate because $k \approx 2000/731 \approx 2.7$, so no further segmentation is needed on these levels. However, there are too many genres in the input dataset ($k = 149$), which would be too many arms to pull for the agent. The solution is to further segment the genre level into a more general level, using the KMeans algorithm on the sound attributes.

3.2 Explicit Segmentation

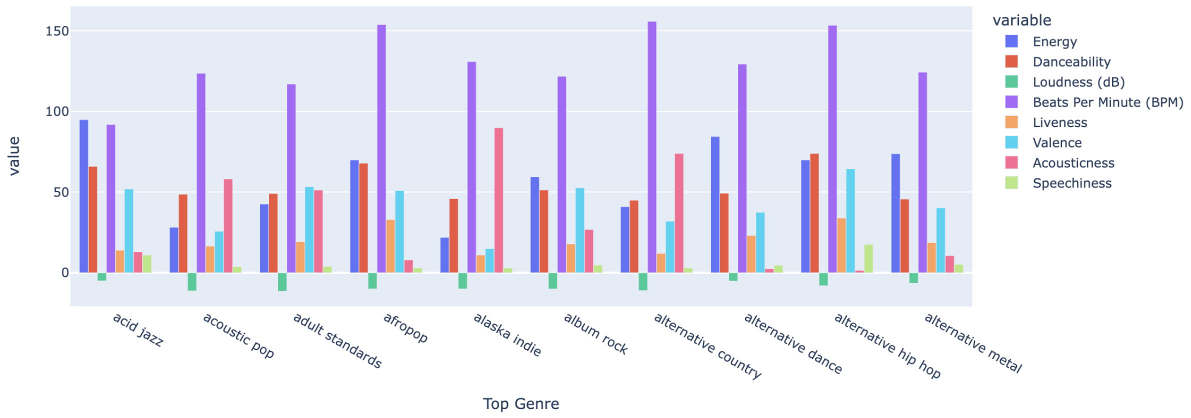


Figure 2: The sound patterns of genres

Figure 2 shows that there exist sound patterns that distinguish different genres, and we hope that we can utilize these patterns to create clusters that group together genres with similar characteristics.

We run a pipeline that first standardizes the attribute values, and then performs KMeans on the genres. In this process, there is a hyperparameter that needs to be chosen somewhat arbitrarily, namely the number of clusters to create. Our choice is $NCluster = 20$, which results in the most balanced structure.

The largest cluster contains around 25 children, which is a reasonable number of possible actions to choose from. However, there are a few degenerate clusters that contain only one child genre. This does not really create problems for us, because we can simply allow the degenerate clusters to skip to the next level without the possibility of exploring during learning.

To do a quick sanity check, we also examined the member genres in each cluster, for example, the following cluster looks like a reasonable group to us.

```
 [acid jazz, basshall, belgian pop, boy band, detroit hip hop,
dutch hip hop, electropop, g funk, indie anthem-folk, reggae]
```

As a proof of concept, we only created one intermediate level in the data tree by segmentation. In practice, it might be possible that one needs to create many intermediate levels to achieve the desired level of granularity in the hierarchy.

3.3 Initialization

Before beginning to learn, we must first initialize data structures that will be used during learning, in particular Q, arm, and $rewards$.

Whereas in traditional K-bandits, there is a global Q and arm, in our hierarchical case we must keep individual Qs and arms for each cluster, genre, and artist, and the size is exactly the number of children this particular node has. We used hashmaps to keep track of which sub-branch is associated with which Q/arm.

To simulate $rewards$, we considered a particular case where the music listener has a favorite artist, and a favorite song made by the favorite artist. All of the artist's song has $reward = 1$, except the favorite song which has $reward = 2$. All other songs have $reward = 0$.

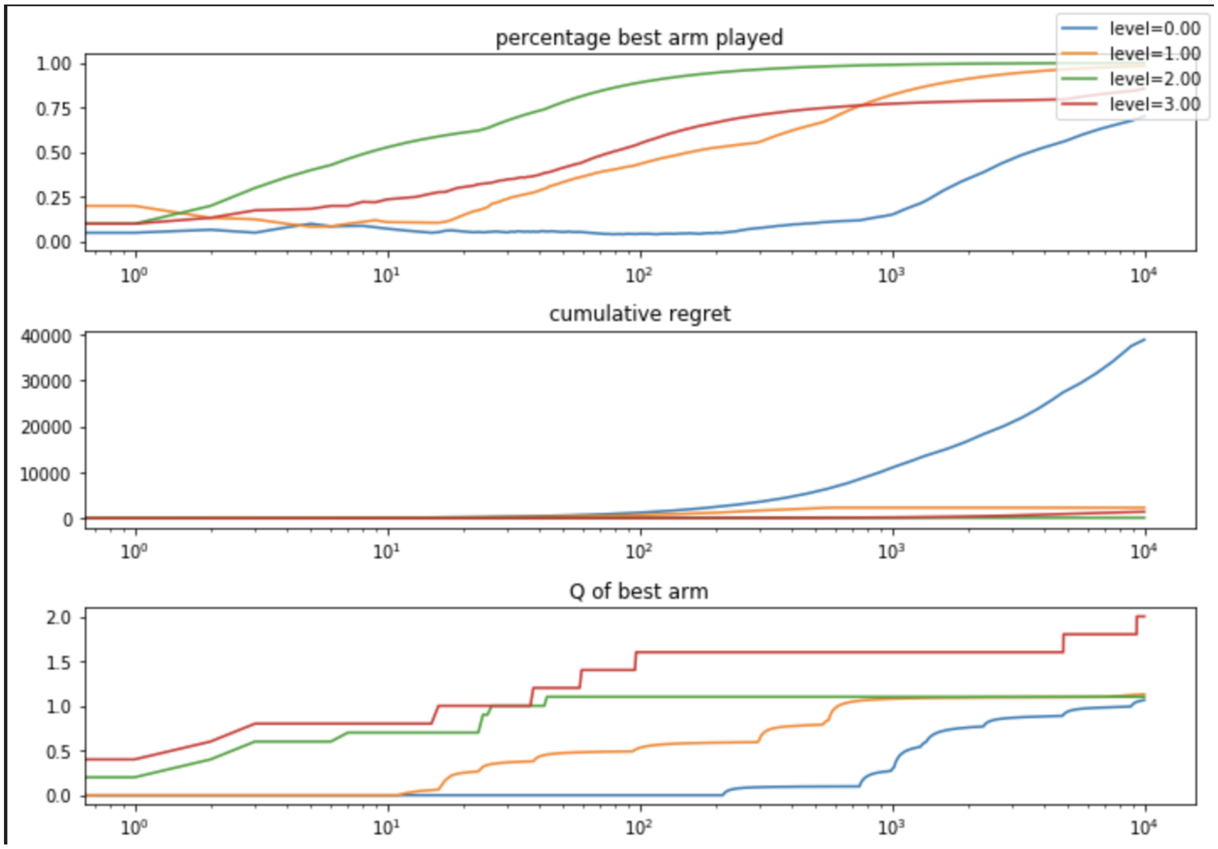
3.4 Learning

During the learning process, the agent always starts at the root node. The agent recursively invokes a sub-branch to compute the reward for it, updating the Q and arm values of all nodes encountered along the way, until the recursion reaches the leaf level, which finally propagates the song's reward back up the calling stack of functions.

For example, if we initialize our favorite artist to be "Ed Sheeran" and our favorite song to be "Perfect", then we can run a ϵ-decreasing algorithm on the dataset, and get the trace in Figure 3.4. The plots are the percentage of best arm played, cumulative regret, and the value of best arm respectively. The different colors represent different levels in the data tree (for the sake of clarity, we omitted all sub-branches that do not contain any rewards).

It can be observed that all arms are able to eventually converge to the optimal value, i.e., they are able to find our favorite artist and favorite song. If we follow the agent's decisions, we will first obtain the correct cluster, and then the correct genre (pop), the correct artist, and the finally correct song.

Notice that for this system to be a recommender system, all we need to do is rely on the agent's exploration to recommend new songs based on the values that it has learned. In our current example, since the agent has learned that our favorite song belongs to the "pop" genre, it is more likely to explore that genre than other genres, based on our rule of hierarchical exploration. The same logic applies to clusters.



4 Conclusion

In this project, we presented a novel approach of using hierarchical bandits to perform recommendation tasks, and applied the technique to a music dataset, getting meaningful results. The chief contribution of this project is to formulate a generalization of traditional MAB that allows bandit algorithms to be run on huge sample spaces efficiently.

However, there are a few shortcomings of this approach. First, the segmentation of the input data into a tree structure is somewhat arbitrary and relies on well-chosen hyperparameters. Second, the learning process suffers from some degree of inefficiency, because $\log_k N$ sub-decisions have to be made for one decision. Third, the propagation of rewards in deep trees can be ineffective (in Figure 3.4, the blue curve converges most slowly and has the largest regret, because it is furthest from the leaf). How to overcome these shortcomings can be topic for future research.

References

[1] https://www.kaggle.com/datasets/iamsumat/spotify-top-2000s-mega-dataset