A Hybrid Approach to Recommend Long Tail Items

Diogo Vinícius de Sousa Silva
Federal University of Bahia
Salvador, Bahia, Brazil
diogovss@ufba.br

Frederico Araújo Durão
Federal University of Bahia
Salvador, Bahia, Brazil
fdurao@ufba.br

ABSTRACT

Techniques in recommendation systems generally focuses on recommending the most important items for a user. The purpose of this work is to generate recommendations focusing on long tail items, and then to conduct the user to less popular items. However, such items are of great relevance to the user. Two techniques from the literature were applied in this study in a hybrid way. The first technique is through markov chains to calculate node similarity of a user item graph. The second technique applies clustering, where items are separated into distinct clusters: popular items (short tail) and non-popular items (long tail). Using the Movielens 100k database, we conducted an experiment to calculate the accuracy, diversity, and popularity of the recommended items. With our hybrid approach we were able to improve the recall by up to 27.97 % when compared to the markov chain-based algorithm, which indicates greater targeting to long tail products. At the same time the recommended items were more diversified and less popular, which indicates greater targeting to long tail products.

KEYWORDS

Recommender System; Long Tail; Graphs; Markov Chain; Clusterization.

1 INTRODUCTION

Most of the methods used by Recommendation Systems (RS) tend to recommend the most popular items to users. Since the majority of users are interested in an item, the likelihood is that a new user will also be interested in that item [8]. By following this logic, it is natural that less popular items are less recommended and consequently less consumed. Usually, the main companies focus their sales on these products envisioning better logistic. If we imagine a company with physical stock, it is easy to understand that it is much cheaper to put the best-selling products on the more evident shelves. However, with the advent of virtual stores the cost of organizing products on shelves is non-existent. In the context of virtual stores has raised the term "infinite-inventory" [1], where products in evidence can be selected according to the preference of each online user. This virtual user will not necessarily have the same preference as other users.

The term "long tail" refers to the set of products not commonly consumed by users [1]. Usually, these products make up the bulk of store stock with low demand. Only a small amount of products contribute for the majority of sales. In contrast, most products (which would be in long tail) are responsible only for the minority of sales. According to Paretto's rule [11], 80% of the consequences come from 20% of the causes, that is, 80% of sales would be concentrated in only 20% of the products from the stock. Long tail items are those less popular items, while the other items, i.e. the most popular, will be called "short tail" items.

The long tail phenomenon can also be seen as a way to increase company profits. Generally high popular products are quite marketed by several other companies and of course the competition for sales is great. Since there is a great demand for these products, the price tends to be the lowest due to competition. Therefore, the products profit rate becomes quite low. Considering items with low demand, it is possible to predict a higher profit margin for interested users. The users will be more motivated to pay a higher price due to the low product availability. Another effect of exploring long tail products is the so-called "one-stop shopping convenience" effect. A store that offers long tail products and also popular products delivers an additional convenience for their customers since you find everything you need in one place.

In this context, this work proposes a hybrid technique for improving the recommendation of long tail products. Our approach uses a structure that represents the items and users in a bipartite graph. Then, we determine the proximity through Markov chains using the Hitting Time algorithm. In addition, we also combine the use of this algorithm with a technique of splitting and clustering in order to improve the accuracy of long tail items recommendations. Finally, we indicate the long tail items for recommendations, thus turning the algorithm more assertive and improving the recommendations diversity.

The remainder of the work is organized as follows: Section 2 presents the related works. The Section 3 addresses the proposed long tail recommendation. Section 4 details the experimental evaluation of the proposed technique and discusses the results of the experiments. Finally, Section 5 traces some conclusions and future work.

2 RELATED WORK

Yin et al. [12] has developed four variations of an algorithm for long tail item recommendations. The basic algorithm of the proposal is the Hitting Time, where the users and items are represented in a disjunct, indirect and bipartite graph. From this graph an adjacency matrix is obtained, as shown in Figure 1. The edges of the graph are weighted and represent the relevance of a user's connection to an item, that is, a user's rating for that item. To calculate the proximity of unrecorded items, the author calculates the Hitting Time using a type of Markov chains called Random Walk [2]. Using Random Walk, the probability of a user reaching an item not evaluated by him is calculated. The higher the probability, the lower the Hitting Time, and therefore the item should have higher priority in the

In: XVIII Workshop de Teses de Dissertações (WTD 2018), Salvador, Brasil. Anais do XXIV Simpósio Brasileiro de Sistemas Multimídia e Web: Workshops e Pôsteres. Porto Alegre: Sociedade Brasileira de Computação, 2018.
© 2018 SBC – Sociedade Brasileira de Computação.
ISBN 978-85-7669-435-9.

recommendation. Transition matrices are derived from a probability matrix. The probability matrix is obtained from the adjacency matrix shown in Figure 1. Shang et al. [10] perform a study of a custom recommendation model using collaborative filtering and ternary relationships, through tripartite graphs, representing users, items, and tags. It proposes a new measure for user similarity based on user tags and preferences. The similarities are calculated using a diffusion-based process and finally compared with recommendations calculated based on similarity of the cosine. Johnson et al. [4] carry out an extension of the work of Yin et al. [12]. Johnson combines the algorithms used by Yin, adapting to tripartite graphs, shown by Shang. Johnson combined the Yin approach with the study of Shang to generate recommendations by collaborative filtering.

Park and Tuzhilin [7] suggest an approach based on splitting and clustering item set. The item set is divided between short tail and various parts of the long tail. Thus, recommendation of short tail items are made based on the individual scores of each item. In the case of the long tail items recommendation, the clusters scores are grouped in each part of the long tail. Park in [6] evolves the previous work by clustering the items based on popularity. The author presents a technique called *adaptive clustering* for recommendations of items according to their popularity. It is possible to define the size and quantities of the clusters in an adaptive way according to the state of the dataset, focusing on items long tail. As a result you get lower error rates and improved performance.

The approach presented in this work uses Hitting Time combined with a technique for item clustering. Our proposal will be based on the Park's approach [7]. We will group the long tail items into clusters according to the average ratings of each item and use this score as weight. The weight represents an additional variable in the Hitting Time algorithm. The main objective is to increase the emphasis on the recommendation of items located in long tail improving the accuracy.

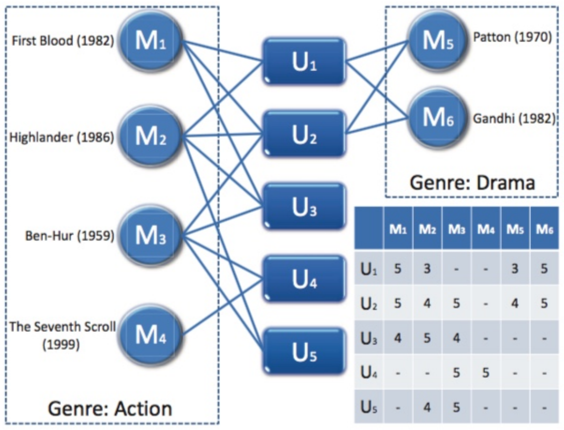
3 PROPOSED SOLUTION

In this section we will explain our hybrid approach composed of Hitting Time algorithm and a clustering technique will turn the recommendations more focused on long tail items.

3.1 Hitting Time

In the Hitting Time algorithm [12] the set of users U and items M are represented in a split-graph and have their corresponding adjacency matrix. Figure 1 illustrates an example of the graph with nodes that could be items or users. The edges of these nodes carry a weight that reflect the user's rating for that item that the edge binds to. When there is no edge connecting two nodes, the weight is 0 (zero). As an example, we have the user node U_1 that has a connection with the item node M_6. In the matrix of Figure 1 the weight of this edge is 5. In other words, the rating of such a user for item M_6 has a value of 5. The weight of an edge is represented by $a(i, j)$.The variables i and j are the nodes of the graph in an array $A = (a(i, j))_{i,j \in V}$. The variable V represents the set of vertices (nodes) of the graph.

Aiming at calculating the proximity of two nodes in the graph, the algorithm uses a specific type of Markov chain, called random



	M_1	M_2	M_3	M_4	M_5	M_6
U_1	5	3	-	-	3	5
U_2	5	4	5	-	4	5
U_3	4	5	4	-	-	-
U_4	-	-	5	5	-	-
U_5	-	4	5	-	-	-

Figure 1: Representation of users and items through a bipartite graph and its respective adjacency matrix [12].

walk. A random walk has a current state (a node in the graph) and with each evolution in time the state is changed, that is, visiting other nodes of the graph. Through the hitting time this path is driven by the weights of the edges and the random walk ends its walk when it arrives at the target node. The algorithm is based on a probability matrix that is computed from the adjacency matrix. Finally, the edge weights are given by Equation $p_{i,j} = P(s(t + 1) = j|s(t) = i) = \frac{a(i,j)}{d_i}$, where $d_i = \sum_{j=1}^{n} a(i, j)$.

Taking into account the probability matrix, a transition matrix is calculated for each chain state until the destination state is reach. Next, the hitting time value may be obtained through of the $H(q|j) = \frac{1}{p_{j,q}} = \frac{\pi_j}{p_{q,j}\pi_q}$, where $\pi_i = \frac{\sum_{j=1}^{n} a(i,j)}{\sum_{i,j=1}^{n} a(i,j)}$.

The Hitting Time algorithm is based on the time-reversibility property to guide recommendations with long tail items. This property indicates that the paths are not symmetric. The probability that a node A reaches a node B is different from the probability that node B reaches node A [2]. The algorithm calculation takes the inverse path to calculate the value, that is, from user node to item node. In this way all items, including long tail items, are equally considered for recommendations. Thus, long tail items tend to be more recommended. This fact would not happen if the calculation began in the user-item sense, since there would be more paths that would link to the most popular products.

3.2 Clustering

We divide the set of items between short tail items and long tail items. At this point we use the Paretto's rule [11] as a parameter to separate the most popular items from the less popular ones. All long tail items are clustered considering the score of each item.

Those items with higher scores will have higher priority in the recommendation and will have a greater weight. Such items impact on the decrease in the Hitting Time value. The items with smaller scores are grouped in clusters that will be used to ponder the value of Hitting Time, increasing its size. The value resulting from the

Table 1: Clustering of the item dataset related to the average rating (score) and its respective adjustment factor.

Score Item (S)	Cluster	Adjustment Factor (%)
1 <= S < 2	A	+20
2 <= S < 3	B	+10
3 <= S < 4	C	-10
4 <= S <= 5	D	-20

Table 2: Ordering items based on the Hitting Time algorithm.

Item	Hitting Time	Priority for Recommendation
Titanic	12	1º
Little Dorrit	13	2º
Batman	14	3º
Simple Simon	15	4º
Black	16	5º

application of the adjustment factor to the value found through the algorithm Hitting Time will be called Hitting Time Clustered *(HTCL)*. The value of the adjustment factor that will increase or decrease the Hitting Time is described in Table 1.

To calculate the Hitting Time Clustered *(HTCL)* we first need to calculate the value to be added. This value may be negative or positive, depending on the cluster in which the item was clustered (see Table 1). This additional value will be added to the previously calculated Hitting Time value. The equation $HTCL(q|j) = H(q|j) + \frac{AF \cdot H(q|j)}{100}$ shows this calculation, where AF means the adjusting factor based on Table 1, $H(q|j)$ means the Hitting Time of an item j for a user q and $HTCL(q|j)$ means the Hitting Time Clustered.

3.2.1 Algorithm In Action. In order to illustrate the algorithm operation, Table 2 presents a list with 5 items recommended for a given user. In our context the items presented are examples of movies. As stated in the previous section, the smaller the hitting time the closer the user is to the item, consequently the item will have higher priority in the recommendation. In Table 2 the items are presented in order of relevance, according to the hitting time.

Now, let's look at the distribution of items in a chart that represents long tail items and short tail items. Figure 2 represents the point of the set item where splitting is performed and how the items are clustered. In our example, there are 2 items that are presented in the short tail (movies "Titanic" and "Batman") and 3 items in the long tail (movies "Little Dorrit", "Simple Simon" and "Black"). For each item in the long tail, its score is shown, i.e. the average of the received ratings. The clustering algorithm will perform similarity classification based on the score (according to Table 1). In this way the movies "Little Dorrit" and "Simple Simon" will be considered similar and will be together in cluster C and the item "Black" will be alone in cluster D. The movies "Titanic" and "Batman" as they are located in the short tail will not be clustered.

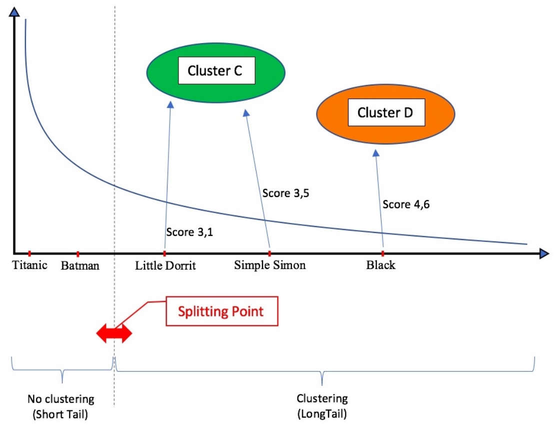


Figure 2: Graphic illustration of splitting and clustering in item dataset.

Table 3: Comparing values between Hitting Time and Hitting Time Clustered.

Item	Hitting Time	Hitting Time Clustered
Little Dorrit	13	11,7
Simple Simon	15	13,5
Black	16	12,8

Table 4: Result after dataset clustering.

Priority for Recommendation	Previous Ranking	Current Ranking
1º	Titanic	Little Dorrit
2º	Little Dorrit	Titanic
3º	Batman	Black
4º	Simple Simon	Simple Simon
5º	Black	Batman

Table 3 lists the 3 items that are located in long tail. For each of them the calculation result is shown by the HTCL approach, based on the adjustment factor shown in Table 1. The order of relevance for the recommendation of the items will no longer be the Hitting Time (HT), but the Hitting Time Clustered (HTCL), instead.

In the Table 4 we see the relevance items order changing after the clustering presented here. We can observe that movie "Titanic" was previously the most relevant recommendation for the user. After the application of clustering we see that movie "Little Dorrit" becomes the most relevant. Notice that in Figure 2 movie "Titanic" is in short tail and movie "Little Dorrit" in longtail. That is, long tail are more prioritized rather than short tail items by just adding the clustering technique. The logic of the Hitting Time algorithm remained the same. We can also observe that the other items also changed position in the ranking of recommendations. Movie "Black" (present in long tail) has increased in the ranking and item "Batman" (present in short tail) fell in the ranking. The item "Simple

Simon" remained in the same position, despite being in the long tail, showing that clustering does not always prioritize the items that are in the long tail.

With such approach the trend is that long tail items with better user ratings take higher priorities. In addition, items that are at the end of the tail can also be recommended more easily and more assertively, since thay have a good score.

4 EVALUATION

4.1 Dataset

The dataset used in this study was the Movielens100k dataset [3]. The Movielens dataset has approximately 1,682 movies and contains 100,000 ratings on a scale of 1 to 5 scored by around 943 users. The density for the rating matrix is 6.30%, a sparse matrix, which means that most users have not seen most movies. The dataset also includes other information that was not used in this study, such as: age, gender, user occupation, and category of movies.

Each user has rated at least 20 movies. The data was collected through the MovieLens web site (http://movielens.umn.edu) during the seven-month period from September 19th, 1997 through April 22nd, 1998. This data has been cleaned up - users who had less than 20 ratings or did not have complete demographic information were removed from this dataset.

4.2 Metrics

4.2.1 *Recall.*

To calculate the accuracy of the proposal, we use a metric called Recall@N, previously adopted by [12]. Recall is the index that indicates the amount of items of interest to the user that appear in the list of recommendations. This index ranges from 0 to 1. The closer to 1 better the recommendation. The metrics idea is to select a user favorite item and include it in a set of randomly selected items. After that, the algorithm ranks all set items and then checks whether the user's favorite item is displayed in the top@N.

For the recall application we split the dataset into two subset. The first, called test set (or estimation test) was used to randomly select items located in long tail and evaluated with 5 stars. The second subset was used to perform the tests and validation of the generated recommendations. For each item M present in the training subset, we select other items that have not been evaluated by the same user who evaluated item M. Let's call this training set as B. From there we execute the algorithm and rank all items not evaluated together with the item M. A test case, as described above with item M, is performed several times. In our experiment we varied the number of test cases together with the number of items present in training set B. A number of 500 test cases were executed. Each experiment used a training set B of 100 items.

To calculate the Recall@N we define hit@N like a test case. It is counted how many times the item M appears inside the top@N results, as shown in the following equation $Recall@N = \frac{\sum hit@N}{|L|}$, where $|L|$ is the number of test cases. This way, the higher the recall result, the greater the accuracy of the tested algorithm. Which represents that there are more items of user preference being returned in the top@N results.

4.2.2 *Diversity.*

The metric diversity [5] was used to obtain the degree of distribution of long tail items that are recommended. With a high diversity the tendency is that long tail items are recommended to the users, that is, the recommendations suffer little influence from very popular items. Such influence causes the discovery of new items hitherto hidden in the long tail.

To calculate diversity we use the same calculation shown in [12]. This metric calculates for a given set of users the top@K items to recommend. Let's assume that we will use a set of 20 users and for each of them a top@10 ranking, that is, 10 recommended items. In this case we would have a total of 200 recommended items for all users. To calculate the diversity we check how many repeated items are counted only once and then calculate the proportion to the total, as shown in $Diversity = \frac{|\bigcup I_u \in I|}{|U| \cdot top@N}$, where I_u is the set of single items recommended for all users. The I element means the set of items in the dataset, U represents the set of users and $top@N$ means the number of items recommended for each user.

In our experiments we fixed a set with 200 users and for each user varied the amount of recommended items in top@10, top@20, top@30, top@40 and top@50.

4.2.3 *Popularity.*

The metric popularity represents the quantity of recommended long tail and short tail items. An analysis of this metric together with the others result in a more careful analysis regarding the algorithm performance. This metric calculates the frequency of an item according to the amount of ratings it holds because of the other ratings of dataset [12].

Considering that our dataset has 100,000 ratings, we calculated the popularity of the top@10, top@20, top@30, top@40 and top@50. For each of them we selected 200 different users. The calculation was based on the average popularity of items in each user's ranking. And for each user the average was calculated to arrive at the final value. That is, we count the amount of evaluations received for each item. And then we relate to the total amount of dataset evaluations, as shown in $Popularity = \frac{R_u}{\sum |R_d|}$, where $R_u = \frac{\sum |R_r|}{|U| \cdot top@N}$, where R_d represents the rating set of the entire dataset. The number of ratings is about 100,000 and the U element represents the set of users selected for the popularity calculation. The *top@N* element means the amount of recommended items for each user belonging to the U set. In Equation above, R_u represents the rating rate already normalized according to the number of users and recommended items.

The higher the result, the greater the popularity of a particular item. Thus, to have a good result of the algorithm in relation to the recommendation of long tail items, it is interesting that this metric has a low value, that is, recommend little popular items to users. Both the *diversity* metric and the *popularity* metric can measure how far the recommendation is directed to long tail items.

4.3 Baselines

To analyze the effectiveness of our Hitting Time Clustered approach (HTCL), we performed a comparison with 3 different baselines, namely:

- **Hitting Time (HT)** - The algorithm proposed by Yin [12] is presented in Section 3.1.
- **Hitting Time + Clustering All Dataset (HTCA)** - The Hitting Time Algorithm plus clustering similar to our approach. The difference here is in the lack of splitting of

the dataset. That is, there was no separation of items between long tail and short tail and then clustering occurred throughout the dataset.

- **Hitting Time + Clustering Short Tail Dataset (HTCS)** - Also similar to our approach, however, the splitting in the dataset was done in the opposite way. Instead of clustering the items located in the long tail, in this baseline we cluster only the items present in the short tail of the dataset.

4.4 Results

4.4.1 Recall Measurement. Figure 3 shows the performace of our approach (HTCL) compared to the baselines described in the Section 4.3. In the execution presented in the graph were ranked 100 items and then calculated the recall for 500 test cases. We can observe that of all baselines, our approach obtained the best values for the recall in all *top@N*.

Note that when we used only HT, the results were better than any of the other baselines that use clustering, with the exception of HTCL. For example, at the top@10 point, the value in baseline HT is 0.086, when we change to our HTCL approach the value was 0.094. In top@35 the HTCL outperforms HT in 27,97%. When using HTCA and HTCS, the result was worse than HT in all top@N. With the HTCS we had the worst result of all. In top@5 HT outperforms in HTCA 31,25% and HTCS in 34,38%. That is, clustering only the short tail will make the recommendations worse than not clustering anything. The best results for the HTCL confirmed the effectiveness of our approach, because the clustering of items in the long tail tend to have a recommendation with better recall.

Other experiments were also performed by varying the number of test cases and the number of ranked items. However the results were similar, following the same order as shown in Figure 3. That is, the best approach was HTCL followed by HT, HTCA and finally HTCS.

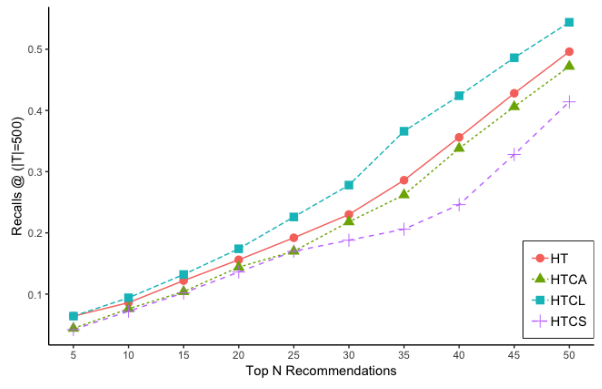


Figure 3: Recall of the top@N items in 500 test cases.

4.4.2 Diversity Measurement. The results of the measurement of diversity were also positive for HTCL. Figure 4 presents the results for an amount of 200 random users when generating top@N recommendations. In this measurement, although the increase is low, it is noticeable mainly when the HTCL recommends from top@30 items. Note that at the top@30, the HTCL approach presents 0.0347, versus 0.345 of the HT approach. That is, the items recommended by our proposal have a greater diversity of items. This approach proved to be the best, followed by HT. And following the same order of the recall, the third best was the HTCA and finally the HTCS.

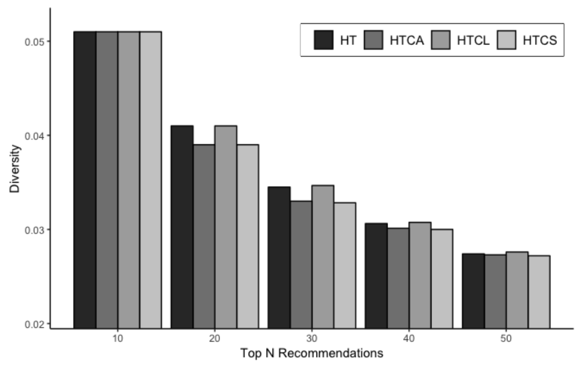


Figure 4: Diversity on Movielens 100k using 200 random users.

4.4.3 Popularity Measurement. In terms of popularity measurements, there was no unanimity. Analyzing Figure 5 we see that the results were quite close to all baselines. The HTCA and HTCS approaches were the ones with the lowest popularity ratings. Our approach got the same result as HT up to the top@20. From then on it managed to overcome HT. See that in top@30 the popularity index in HTCL is 0.3119 and the index in HT is 0.3122. That is, our recommendations tend to suggest less popular products.

Low popularity is not bad for our purpose, since the recommendation of long tail items needs to be more focused on niche products and not among the top sellers. In short, the HTCL approach improves diversity and returns less popular items, which is good, as we are focusing on long tail recommendations. In addition, the good results in diversity and popularity were reached without harming the accuracy, since in the recall method our approach obtained the best results.

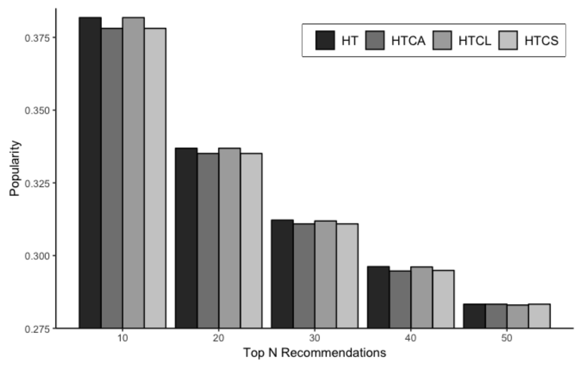


Figure 5: Popularity on Movielens 100k using 500 random users.

4.5 Discussion and Points of Improvements

The approach proposed in this work (HTCL) obtained the best results in the 3 tested metrics. The recommendations tend to be more focused on longtail items, since our approach uses the Hitting Time algorithm together with the clustering technique. The results of baselines HT and HTCL when compared in metrics diversity and popularity show the superiority of our approach. Until reaching the top@20, the two baselines do not show significant differences. The top results of the HTCL approach are evident from top@30, when the HTCL baseline begins to recommend less popular and more diverse items. The results of the other two baselines HTCA (clutter of the entire dataset) and the HTCS (clustering only of short tail) show that clustering if performed in any way can lead to worse results. That is, the clustering performed in these baselines worsened the results already obtained without any type of clustering (baseline HT).

The items diversity was improved and there was a decrease in popularity, but the relevance of the recommendations was not affected. The recall metric allowed us to monitor the accuracy of the recommendations. The HTCL approach obtained the best results at all top@N (see Figure 3). The other baselines that performed the clustering differently, worsened the results of the HT approach. This result has showed the importance of defining a good strategy when clustering a dataset.

Only the variable "item score" was used in the clustering step. Only with this variable the results have already improved, confirming the feasibility of applying different techniques to reach the same goal. Besides the score, there are other variables that can be taken into account in the calculation of similarity, such as: category of the film, producer, and cast. User clustering can also be done through his/her profile data, including age, occupation, gender, among others. These variables are already present in the dataset used in this work and will be the subject of new experiments.

Other techniques can be used in conjunction with a base algorithm and improve recommendations to further explore the long tail. Our approach used Hitting Time as the base algorithm, but other algorithms can be adopted in conjunction with the various clustering techniques. Another possibility is to use, together with clustering, other techniques such as *probabilistic CF algorithm (IRM2), multimodal similarity and multi-objective evolutionary algorithm (MORS)*, just to name a few.

A limitation is related to the size of the dataset and its sparsity. Datasets with different dimensions and sparsities could generate other results. A deeper analysis could bring new information. The other limitation would be the use of a single domain. We have applied a dataset that aggregates movie ratings, called MovieLens. Analyses in other domains could give different results showing specificities of each tested domain.

5 CONCLUSION

In this work we performed a study to generate recommendations of long tail items. The combination of the Hitting Time algorithm with a clustering technique was adopted in order to give more visibility to long tail items. Using the 100k ratings of Movielens database, we conducted an experiment to calculate the recall, diversity, and popularity of the recommended items.

The results indicate that two techniques used together may improve the result of a first technique used alone. This procedure should be done carefully, otherwise the effect may be negative. Our proposal presented satisfactory results by focusing on longtail items. We have increased the diversity of recommendations. There was a decrease in popularity of the recommended products. At the same time, the relevance of the recommendations (measured by the recall metric) was also higher compared to the tested approaches. The positive results point out possibilities for retail companies that aim at increasing the profit of their businesses. Since the profit from selling long tail items tend to be larger than short tail items. Focusing part of the sales for these products will bring greater financial returns. This is evidenced by the lower competition for such products. In addition, customers of niche products are usually more loyal and are more willing to pay a higher amount to acquire it.

As future work we intend to extend this work using more variables in the clustering of items, such as age of the user, category of the item, among others. We also intend to adopt other clustering techniques as proposed by [9]. Another possibility is to try out the other algorithms proposed by [12], combining them with the presented clustering technique. Other metrics can be added such as Time, Precision, and F-measure. The use of other databases would also help giving more validity to the results, as well making comparisons with other baselines.

REFERENCES

[1] Chris Anderson. 2006. *The Long Tail: Why the Future of Business Is Selling Less of More.* Hyperion.

[2] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. 1998. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications.* Wiley-Interscience, New York, NY, USA.

[3] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4, Article 19 (Dec. 2015), 19 pages. https://doi.org/10.1145/2827872

[4] Joseph Johnson and Yiu-Kai Ng. 2017. Enhancing Long Tail Item Recommendations Using Tripartite Graphs and Markov Process. In *Proceedings of the International Conference on Web Intelligence (WI '17).* ACM, New York, NY, USA, 761–768. https://doi.org/10.1145/3106426.3106439

[5] Matev Kunaver and Toma Porl. 2017. Diversity in Recommender Systems A Survey. *Know.-Based Syst.* 123, C (May 2017), 154–162. https://doi.org/10.1016/j.knosys.2017.02.009

[6] Y. J. Park. 2013. The Adaptive Clustering Method for the Long Tail Problem of Recommender Systems. *IEEE Transactions on Knowledge and Data Engineering* 25, 8 (Aug 2013), 1904–1915. https://doi.org/10.1109/TKDE.2012.119

[7] Yoon-Joo Park and Alexander Tuzhilin. 2008. The Long Tail of Recommender Systems and How to Leverage It. In *Proceedings of the 2008 ACM Conference on Recommender Systems (RecSys '08).* ACM, New York, NY, USA, 11–18. https://doi.org/10.1145/1454008.1454012

[8] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to Recommender Systems Handbook. In *Recommender Systems Handbook.* Springer, 1–35.

[9] Jia Rongfei, Jin Maozhong, and Liu Chao. 2010. A new clustering method for collaborative filtering. In *2010 International Conference on Networking and Information Technology.* 488–492. https://doi.org/10.1109/ICNIT.2010.5508465

[10] Ming-Sheng Shang, Zi-Ke Zhang, Tao Zhou, and Yi-Cheng Zhang. 2010. Collaborative filtering with diffusion-based similarity on tripartite graphs. *Physica A: Statistical Mechanics and its Applications* 389, 6 (2010), 1259 – 1264. https://doi.org/10.1016/j.physa.2009.11.041

[11] Kazuhiro Yamashita, Shane McIntosh, Yasutaka Kamei, Ahmed E. Hassan, and Naoyasu Ubayashi. 2015. Revisiting the Applicability of the Pareto Principle to Core Development Teams in Open Source Software Projects. In *Proceedings of the 14th International Workshop on Principles of Software Evolution (IWPSE 2015).* ACM, New York, NY, USA, 46–55. https://doi.org/10.1145/2804360.2804366

[12] Hongzhi Yin, Bin Cui, Jing Li, Junjie Yao, and Chen Chen. 2012. Challenging the Long Tail Recommendation. *Proc. VLDB Endow.* 5, 9 (May 2012), 896–907. https://doi.org/10.14778/2311906.2311916