Replication Assignment

LEAH AJMANI, CHEN HU, and RUIXUAN SUN

In this paper, we replicate and extend Ludewig et al's *Effective Nearest-Neighbor for Music Recommendations*. The original system was a submission to the RecSys18 Challenge sponsored by Spotify, which was an open call for solutions to the playlist continuation task. We report on the difficulties and result discrepancies we encountered while attempting to reproduce Ludewig et al's work. We then extend their original system in four ways: (1) We evaluate the recommender on different size of datsets. (2) By adding an SVD-based approach to their hybrid recommender, we find that our SVD algorithm doesn't beat the best individual algorithm in the original paper, but improves the performance of the combined recommender. (3) We add pearson similarity as parameter tuning (4) We continue adding to Ludewig et al's work by evaluating the system across recall, f1 and RMSE. Our results evidence that the original authors were optimizing for performance on the competition metrics. Namely, R-precision, clicks and NDCG. Finally, we discuss how our experience with replication highlights tensions within recommender systems, such as the disparity between open source code and replicable code, as well as how replication is a powerful tool for justifying design decisions. Moreover, we acknowledge how this task deviates form traditional user-item-ratings based methods. Future work could replicate Ludewig et al's extension with external data, replicate later work on the same task or explore other music recommendation tasks.

ACM Reference Format:
Leah Ajmani, Chen Hu, and Ruixuan Sun. 2021. Replication Assignment. 1, 1 (February 2021), 4 pages.

1 INTRODUCTION

As music consumption shifts from physical albums towards online audio streaming, listening to playlists has become a popular habit for most music consumers in recent years. Automated playlist continuation is a common feature of online music platform. However, effectively suggesting the next track to play is an increasing and important problem for media streaming companies such as Spotify. During 2018 ACM RecSys Challenge, Spotify published the Million Playlist Dataset (MPD) for all the participating teams to study the automated playlist continuation problem. Several peer reviewed papers have been published by different teams to demonstrate their approaches and algorithms [4]. Both traditional methods, such as collaborative filtering and state-of-the-art methods, such as neural networks, were submitted to the challenge. Even though fashion deep learning methods have proven its advantages, traditional methods such as k-NN and collaborative filtering still output competitive performance. These methods can also be less time and resource intensive compared to their deep learning counterparts. Therefore, it's still worthwhile to investigate and extend the collaborative filtering methods in the scope of the Spotify MPD challenge.

Furthermore, reproduction and replication of recommender systems has become increasingly important to the field. The aim of replication is to both confirm the conclusions drawn by authors in the original paper and see if new ideas or approaches are applicable. Unsuccessful reproduction can reshape the current view of field and indicate the authors may emphasize the novelty or importance of their work too aggressively. Moreover, replication also stimulates the original paper authors to keep their code neat for followers and promote the whole research field to make more reproducible work. In consequence, replication work has become essentially important in current research field.

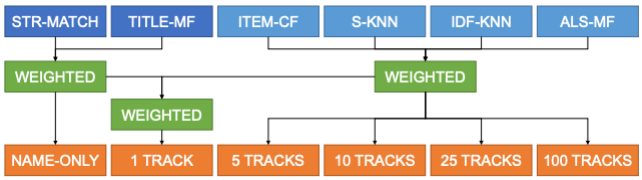

Fig. 1. Composition of Ludewig et al's hybrid recommender [2]

Based on the foregoing reasons, we attempt to replicate and extend one of the papers ***Efficient Nearest Neighbor Music Recommendations*** published by Ludewig et al [2] in RecSys18 Challenge, while other papers are also attempted by a glance. We first repeat their original analysis on the same Spotify dataset to see if we can reproduce the results in their paper as well as forming a basic understanding of the problem. We then explore extensions along four dimensions: (1) data, (2) technical approach, (3) parameters and (4) evaluation. We start by sampling the original dataset to produce a small dataset containing 50k playlists, and then extend the playlist size to be 50k, 100k, 500k and lastly the complete 1m to see if the results shows consistency. Besides, We write our own SVD-based approach and compare with the authors' efficient nearest neighbor algorithm and integrate our SVD model into Ludewig et al's combined recommender. Thirdly, We explore how tuning the original cosine similarity metric affects our results. Finally, we add RMSE, Recall, and F1 score as additional evaluation metrics.

We found a lot of evidence to support Ludewig et al's design decisions such as using cosine similarity and optimizing for performance on rp and NDCG over metrics such as recall, F1 and RMSE. While we were eventually able to make our SVD algorithm work, the individual performance of that was not as good as the algorithms chosen by the original paper. SVD algorithm has intrinsic drawback of adding new user/playlist while the original paper succeed to deal with cold start problem. However, the parameter tuning, evaluation metrics and additional algorithm we add strengthen the robustness of the whole system, thus would gain better performance for future online evaluation.

2 PREVIOUS WORK

In 2018, the annual RecSys challenge focused on music recommendation. Specifically, the challenge of automatic playlist continuation. By suggesting appropriate songs to add to a playlist, a Recommender System can increase user engagement by making playlist creation

Authors' address: Leah Ajmani; Chen Hu; Ruixuan Sun.

XXXX-XXXX/2021/2-ART $15.00
https://doi.org/

easier, as well as extending listening beyond the end of existing playlists [1].

Ludewig et al [2] proposed a nearest-neighbor based system that ranked 7th in the “main" track, which used a provided dataset, and 3rd in the “creative" track, where they were allowed the use of external public sources [4]. Their approach was a hybrid recommender (See Figure 1) that combined the following approaches:

Track-based approaches. For playlists that already included tracks, Ludewig et al used a weighted hybrid of item-based collaborative filtering; idf and session based knn; and also matrix factorization techniques.

Name-based approaches. If a playlist contained no tracks, they used a weighted hybrid of string-matching between playlist and track names as well as title factorization techniques.

Sparse playlists. Many playlists only contained a track or two. While these playlists were not empty, they were too sparse for track-based approaches alone. For these playlists, Ludewig et al used a weighted hybrid of the track-based and name-based approaches.

In essence, Ludewig et al’s system switches between three hybrid approaches –weighted track based, weighted name based or weighted track/name based– determined by playlist length. Not only does this method provide accurate recommendations for established playlists, but it mitigates the cold start problem of initial track recommendation.

3 DATASET EXPLORATION

The dataset provided by Spotify challenge contains 1 million playlists, including titles, tracks, artist information, and other metadata. Besides, Spotify has also provided test dataset containing 10,000 playlists where certain amount of tracks are hidden in each playlist for prediction purpose. For efficiency of our replication, we mainly utilize the randomly selected 50k playlists from the 1 million provided data as our training set to generate the original metrics mentioned in the paper as well as some new metrics added by us. But for the purpose of data exploration, we also explore the impact of size difference on the basic metrics evaluation on the dataset, and we find out that larger number of data would actually induce worse performance on metrics.

Size	RP	Clicks	NDCG
50k	.225	1.62	.439
100k	.226	1.497	.439
500k	.201	1.722	.400
1m	.194	1.872	.387

Table 1. Different sized playlist data and their corresponding metrics results.

In general, lower RP indicates worse precision result, higher number of clicks means that users need to make more average click actions to the right song, and lower NDCG means the top recommended item qualities are not as good for large-sized dataset comparing to the smaller ones. We assume this is caused by the larger variation of training/test split during the offline training session when the dataset size increased. Theoritically, larger dataset would decrease generalization error, which would be good for online training, but we didn’t have a chance to verify so.

4 TECHNICAL EXPERIMENTS

4.1 Initial Replication

At the beginning of replication, we encountered multiple challenges, including shortage of computational resources, packages/installation bugs, obsolete dependencies, poor documentation of the code, and ambiguous taxonomy. To solve those problems, we first tried multiple different remote virtual machines, finally switched back to local laptop to speed up the computation and increase memory allowance. Then we debugged some minor issues with the original code and updated latest library dependencies for the repository. Finally, we tried to read and understand the code logic line by line and figured out the main idea behind the paper.

After all the debugging and preparation steps, we were able to replicate some initial metric results for some of the major methods mentioned in the paper. As shown in 2, we get pretty close results in RP, Clicks, and NDCG metrics on our randomly sampled 50k dataset comparing to the ones calculated by Ludewig et al[2].

	Reported Results			Calculated Results		
Method	RP	Clicks	NDCG	RP	Clicks	NDCG
idf-knn	.208	2.302	.411	.218	2.052	.431
s-knn	.198	2.253	.398	.213	2.034	.422
item-cf	.205	2.571	.379	.215	2.068	.399
als-mf	.184	2.544	.375	.206	2.106	.411
string-match	.097	6.828	.209	.109	5.216	.239
title-mf	.102	7.617	.223	.114	6.704	.248
combined	.214	1.915	.416	.225	1.62	.439

Table 2. Results reported in Ludewig et al [2] compared to results from our initial replication. Both experiments were run on a 50k sample from the original 1M Spotify dataset.

4.2 Parameter Tuning (Similarity)

Ludewig et al’s [2] original work uses cosine similarity for their nearest neighbor based experiments. While they do not elaborate on this decision in their paper, we suspect this is because their knn approaches are item and session based rather than user based. We extended this work by running their knn approaches with a pearson coefficient similarity metric. 3 demonstrates that the pearson cofficient performs worse across RP and NDCG. Our results support Ludewig et al’s decision to optimize their system’s performance by using cosine similarity instead of pearson similarity.

Method	RP	Clicks	NDCG
idf-knn	.209	2.052	.398
s-knn	.198	2.034	.397
item-cf	.178	2.064	.356

Table 3. Results from replicating the original algorithms with pearson similarity instead of cosine similarity.

4.3 SVD Algorithm Addition

After replicating the Ludewig et al’s initial system, we explored (1) how an svd-based recommender system would perform against

the techniques used in the original experiment (2) how the original hybrid recommender system would perform with if it also included an SVD recommender.

In order to integrate an SVD algorithm we used the 'Surprise' python package, which requires a user-item ratings matrix. First, we used Ludewig et al's original method to mimic a ratings matrix in the context of the automatic playlist continuation task. The playlists correspond to the users and the contained tracks are implicit feedback signals to construct a binary user-item "rating" matrix. [2]

We then train our SVD algorithm on the 50k small dataset with the feature dimension implicitly chosen by "Surprise" package. The results of our SVD algorithm is shown in table 4. It can be seen that our algorithm doesn't outperform Ludewig et al's results. The reason might be the intrinsic limitation of SVD algorithm to predict new users/playlists. Since 50k dataset is only a small proportional of the original 1M dataset, most of the playlists in testset haven't been seen by SVD algorithm before, resulting in a cold start problem. Such drawback would make the prediction unstable. And it's believed that with a larger sampled dataset, this problem can be minimized. However, due to the time limitation, we haven't got a chance to test on a bigger dataset.

The last row shows the results we integrate our SVD algorithm into the combined recommender. Even thought single SVD algorithm doesn't show better performance, integrated one do show an improvement in RP, clicks and NDCG compared to reported results in table two. This indicates we add variability in the combined algorithm and thus improve the overall performance.

Method	RP	Clicks	NDCG	Recall	F1	RMSE
diskknn	.218	2.052	.431	.210	.320	.881
sknn	.213	2.034	.422	.202	.310	.887
iknn	.215	2.068	.399	.209	.317	.882
implicit	.205	2.106	.411	.187	.293	.896
smatch	.109	5.216	.239	.108	.172	.939
imatch	.115	6.704	.248	.109	.174	.940
svd	.049	9.524	.136	.049	.087	.974
combined	.217	1.660	.420	.217	.330	.879

Table 4. Results from replicating the small dataset with SVD algorithm.

5 EVALUATION METHODOLOGY

5.1 Original Evaluation

Ludewig et al [2] evaluated their system across the three metrics Spotify used to assess the quality of submissions: R-precision (RP), normalized discounted cumulative gain (NDCG), and recommended songs clicks. "The higher the R-precision and NDCG, the better. However, lower recommended songs clicks indicates better performance." A more detailed description of performance metrics can be found in Zamani et al's [4] survey of challenge submissions.

5.2 Additional Metrics

In addition to the original evaluation metrics, we explored the system's performance across RMSE, recall and F1.

RMSE. We chose to calculate RMSE because the original paper did not use any accuracy metrics. We computed RMSE by comparing the prediction of whether a track is in a playlist continuation to the ground truth from the dataset. Therefore, our RMSE is between 0-1. Our results (see figure 1) demonstrate that none of the recommender algorithms have a significantly high amount of accuracy and don't necessarily align with NDCG performance. For example, implicit has better precision that iknn but worse accuracy. We suspect both of these shortcomings are because none of these algorithms were optimized for accuracy. In the competition instructions it is made clear that entries will only be evaluated on top-n metrics (i.e., RP and NDCG) and click through rate. While no explanation was provided in the challenge description, these metrics make sense given the nature of playlist continuation. In effect, most people do not listen to playlists for an infinite amount of time. Therefore, it makes more sense to optimize a top-n list of tracks that to optimize for overall accuracy across all of the tracks.

Recall & F1. Similarly, none of these algorithms were optimized for overall recall and f1. Therefore, they perform quite poorly on both of these metrics. We can see that these algorithms were optimized for top-n performance when we compare R-precision and ndcg scores to recall and F1 scores.

Low recall performance makes sense for the playlist continuation task because there isn't a high cost associated with false negatives. A Spotify user can remedy a false negative by adding a certain track to either their play queue or the original playlist. Moreover, there isn't a high cost associated with false negatives either. A user can easily skip a song if they feel like it's not relevant. Consequently, low performance on an overall F1 score, which measures the balance between false positives and false negatives, is probably not a major issue. However, a user can get frustrated if they have to skip the first five track recommendations the system offers. Therefore, top-n metrics are more relevant to the playlist continuation task.

6 DISCUSSISON

6.1 Replication is hard (Even with open source materials)

For brevity's sake, we didn't mention the myriad of other papers we read, code we downloaded and datasets we explored before deciding to replicate Ludewig et al's work. However, it's important to note that this paper is part of an underwhelming minority of papers that have made both their code and their dataset publicly available. Furthermore, we believe the only reasons we had access to the code and dataset were because (1) Spotify was responsible for hosting the dataset and (2) public code was a necessary condition for competitors. We hope Ludewig et al's work serves as an example of a successful partnership between researchers and a platform to create publicly available recommender systems.

While we admire the shared code and data, this project demonstrates that publicity is not enough to incentivize replicable work. The majority of our time on this project was spent on the initial replication of this project. As mentioned earlier in this paper, Ludewig et al's code on GitHub relied on deprecated packages, an ambiguous amount of computational resources and sometimes flawed code. Moreover, debugging their code was arduous due to poor code documentation and interpretability. As the RecSys community moves towards a focus on reproducibility, we anticipate work such as

ours can serve as a cautionary tale to not conflate publicity with reproducability.

6.2 Replication can justify implicit design decisions.

Our experiments supported two main design decisions not made explicit in Ludewig et al's paper. First, our results support that using cosine similarity over pearson similarity increases algorithm performance. We assume that this was an intentional decision given that cosine similarity is canonically better suited for item based collaborative filtering.

Second, the recommender's poor performance on RMSE, f1 and recall metrics supported our suspicion that the recommender was optimized for top-n lists. While optimizing for rp, clicks@500 and ndcg@500 makes sense, Ludewig et al's system was built for a competiton– the original authors never mentioned why Spotify was valuing thesse metrics in particular. For example, we only understood the unimportance of false positives when we considered why recall was consistently lower than NDCG.

6.3 Not every recommendation task is a user-item matching problem

We found it really interesting that Ludewig et al didn't use any "user-based" collaborative filtering techniques. Rather, they focused on item and session based methods, only resorting to title based methods when confronted with the cold start problem. This decision makes a lot of sense; the dataset itself didn't have any representation of users. However, this absence of user-based data caused a lot of problems when we attempted to apply an svd algorithm to the dataset. As mentioned in 4.3, the root of these problems was that svd requires a user-item-ratings matrix; something that we could only mimic given the dataset. We hope that as more recommender system packages, such as "Surprise," get built, they incorporate more complex data representations or provide more examples of how to adapt non-traditional datasets to their methods.

7 FUTURE WORK

7.1 Meta re-ranking

Given time constraints on this project, we did not replicate Ludewig et al's [2] submission to the creative track of the challenge. Because the creative track allowed for the use of supplemental data, the authors used track metadata from the Spotify API to enhance their original recommendations. Future work could build off of our initial replication and include Ludewig et al's [2] meta re-ranking algorithm.

7.2 Continuation of challenge

While the RecSys18 challenge ended in July 2018, Spotify continued the challenge on an AI crowd competition platform called AICrowd. The challenge is still open today with the same dataset and task description as the RecSys18 challenge documentation. To date, there have been 508 submissions on the AICrowd challenge with 77 teams competing for "bragging rights." [1] As of Dec 14, 2021, submissions are still bring made every day, implying that competitors are still active. We propose two potential paths for future work based on Spotify's decision to continue the challenge.

First, there are lingering question about why Spotify felt the need to acquire more submissions than those submitted in 2018. Clearly, if work such as Ludewig et al perfectly met Spotify's needs, they wouldn't spend the resources maintaining a crowdsourced competition. Future work could potentially explain this phenomenon by comparing RecSys18 submissions to top AICrowd submissions.

Second, the slew of new recommenders submitted to AICrowd raises questions about reproducability outside of academia. It would be interesting to compare the difficulty in reproducing an open source AICrowd system with Ludewig et al. [2]

7.3 Future Music Recommendation Tasks

In our discusssion, we discussed how music recommendation tasks, such as playlist continuation, go beyond the typical user-item recommendation problem. As Spotify tries to maintain it's market share in the midst of competing products, such as Apple Music and YouTube, it has introduced more complex recommendation features. For example, in September 2021 Spotify released a playlist blending feature which creates a shared, personalized playlist amongst two users [3]. These additional features open up new opportunities to test reproducability or applicability to other domains such as shared personalized playlists on YouTube or shared feeds on social media sites, namely Instagram.

BIBLIOGRAPHY

[1] Spotify million playlist dataset challenge.
[2] Ludewig, M., Kamehkhosh, I., Landia, N., and Jannach, D. Effective nearest-neighbor music recommendations. In *Proceedings of the ACM Recommender Systems Challenge 2018* (New York, NY, USA, 2018), RecSys Challenge '18, Association for Computing Machinery.
[3] Spotify. How spotify's newest personalized experience, blend, creates a playlist for you and your bestie, Sep 2021.
[4] Zamani, H., Schedl, M., Lamere, P., and Chen, C. An analysis of approaches taken in the ACM recsys challenge 2018 for automatic music playlist continuation. *CoRR abs/1810.01520* (2018).