Bundle Generation and Group Recommendation Applied to the Steam Video Game Platform

Francesca Vera
fvera@stanford.edu

Yulian Zhou
zhouyl@stanford.edu

Xiaoyan Wu
xw1@stanford.edu

1 Introduction

The technical challenges and real-world applications associated with recommendation systems are fundamental. Recommendations can be found in social media products, eCommerce sites, publication subscriptions, streaming platforms, and etc. As a result, the sophistication of the technology within these systems has grown enormously and researchers have attempted to develop creative strategies to optimize recommendation for these applications. One such innovation is the possibility of recommending multiple related bundles of items. Another is to provide recommendations for groups of users rather than individuals. One aim of our project is to combine these two ideas by recommending bundles of video games to groups made up of video game players. Another aim of the project is to ensure that we make recommendations to the right groups since group recommendation is compromised when users are randomly selected. We hypothesized that group recommendation performs better for users with similar features, and therefore we apply community detection and user clustering in the user-video game network. We adopted various community detection approaches, especially those working for large graph. We further implement bundle recommendation to user groups based on obtained community information. Our results indicate that community detection and user clustering provide insight into group recommendation.

2 Related Works

2.1 Community Detection in Networks

Video-game players generate huge amounts of data as all their actions within any games get recorded. In the work *A Machine-Learning Item Recommendation System for Video Games* by Bertens et. al[2], they propose an item recommendation system for video games by machine learning. They adapt approaches including extremely randomized trees (ERTs) and deep neural networks (DNNs) to build up two models respectively. In the work *Generating and Personalizing Bundle Recommendations on Steam* by Pathak, et.al[7], the authors generate a personalized game bundle for a group of randomly selected users. They build upon Bayesian Personalized Ranking (BPR), which is trained to estimate ranking of items that are likely to be interacted with. However, both papers fail to take the community feature of network into account, and require *de novo* calculation for an individual user. As the taste of users evolves over time, or when many new users are introduced to the system, the computation required for such recommendation is not trivial. To improve efficacy and accuracy of the video game recommendation, we proposed that pre-computation on the community structure of the game network will facilitate the recommendation process, and community structure of the user network will enable a more robust recommendation to groups of users.

2.2 Group Recommendation Systems

Developing recommenders for groups of users has proven to be popular in research and in product development with these systems having many interesting technical challenges and real-world applications. Many researchers have opted to modify existing techniques developed for individual recommendation systems (e.g. content-based, collaborative filtering etc.) by finding ways to interpret a group's rating or tastes. One common approach is average individual users' item ratings in order to get group ratings.[8] Others utilize rank aggregation, which first relies on a model that predicts individual rank towards the item before aggregating those results for the group.[1] We aim to combine both methods by averaging user ranks to determine group preference. Another important consideration is the way in which groups are chosen. In addition to random selection of users, researchers have considered computing similarity between users (e.g. cosine similarity) in order to smartly select groups.[4] Ntoutsi et al. use a user clustering approach in *Fast Group Recommendations by Applying User Clustering*, in which each user starts in a cluster of their own and the algorithm updates by merging the two most similar clusters at every step.[6] Our strategy for group formation is to analyze the network structure of the data before detecting communities from which groups can be formed, anticipating that this will produce better recommendations than for groups of random users.

2.3 Bundle Generation for Recommendation

Researchers have developed various ways in which bundles are generated. Mohankumar et al. present a "GRAB" algorithm that greedily updates items in bundles and then chooses the bundle that has the lowest ratio of total cost to the individual costs of these desired items.[9] Zhu et al. limits the items that are allowed to appear in a bundle by calculating the "dominance" between two items such that items in the final item set cannot be dominated by more than k other items.[11] We combine the two by greedily selecting from a limited list of items that already exist in bundles.

3 Data Summary

The data we used for our experiments was provided by Julian McAuley, Assistant Professor in the Computer Science Department at UC San Diego, who published several datasets for recommender systems research on his lab's webpage. Under the title *Steam Video Game and Bundle Data*[7], this dataset contains information taken from the Steam video game platform, including 88,310 user profiles, 10,978 game profiles, 59,305 game reviews, and 615 game bundles. According to McAuley's research group, 29,634 users purchased one or more bundles, which suggests that there was a high demand for bundles at the time. This finding is significant because it validates our objective to not only generate appropriate bundles, but also to make bundle recommendations. The average bundle size consists of 5.73 games and approximately 25% of games appear in at least one bundle. Typically, the more popular games appear in the bundles, as approximately 70% of all purchases involved games that appear in bundles.

4 Community Detection

4.1 Game/User Graph Generation

The Steam User-Game network is a bipartite network, which contains user nodes and game nodes, and we perform one-mode projection onto user mode and game mode. Let C_{ij} be incidence matrix

of user-game net:

$$C_{ik} = \begin{cases} 1 & \text{if user } i \text{ plays game } k \\ 0 & \text{otherwise} \end{cases}$$

One-mode network projection to form weighted user network:

$$B_{ij} = \begin{cases} w_{ij} & \#(\text{games}) \text{ that } i \text{ and } j \text{ play} \\ 0 & \text{otherwise} \end{cases}$$

Similarly, weighted game network:

$$D_{kl} = \begin{cases} w_{kl} & \#(\text{users}) \text{ that play both } k \text{ and } l \\ 0 & \text{otherwise} \end{cases}$$

We noticed that the original Steam graph (Figure 1a) is difficult to analyze in practice, because direct projection on the graph will induce the formation of many large cliques. Instead of performing graph contraction, we reconstruct the graph by limiting the users that only play 100 games or less, and games that are played by 1000 users or less (Figure 1b). We also eliminated users and games with 0 degree. The rationale is that (1) users that play too many games may have a broad tastes and games that are played by too many users trend to be popular game, and those users and games may provide less community information. (2) the edge contraction algorithm combines cliques into single "supernodes", and a lot of information regarding individual users and games will be lost.

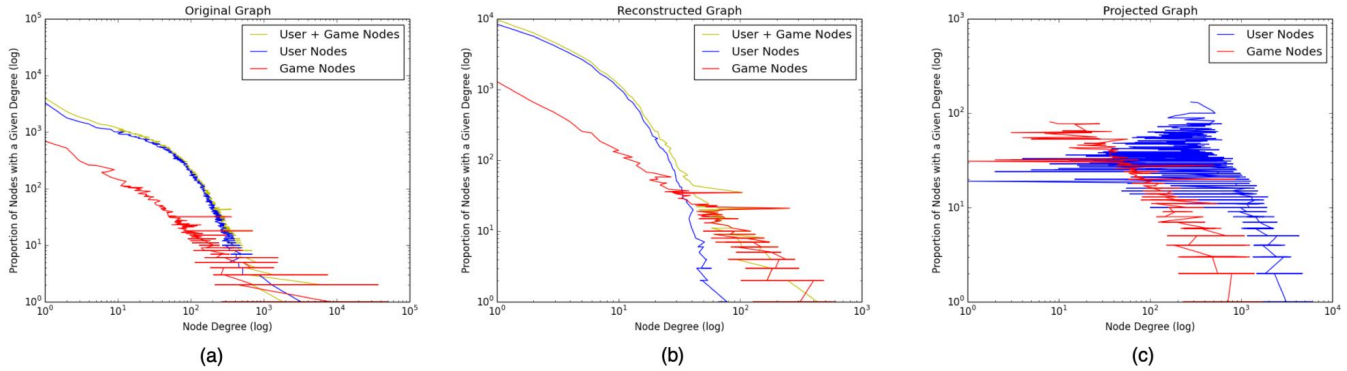

Figure 1: Degree distribution of user nodes and game nodes in original, reconstructed, and projected graph

We perform projection on the reconstructed graph, and generated the User graph and Game graph as undirected edge-weighted graphs with the statistics shown in Table 1. and degree distribution are plotted in Figure 1c.

Table 1: Statistics of Original, Reconstructed and Projected Graphs

	nodes	**edges**	**max WCC size**	**clustering coefficient**
Original graph	99288	5153209	82482	0
Reconstructed graph	44944	225978	44916	0
Projected User graph	38017	14575526	38017	0.532
Projected Game graph	6899	767095	6899	0.535

4.2 Modularity Optimization

After the Steam User and Game graph was generated, we run Louvain's algorithm[3], which greedily maximizes the modularity locally over all nodes. Since modularity measures the density of edges inside communities to edges outside communities, optimizing the value results in the best possible grouping of the nodes in the network. The modularity Q of an undirected graph can be expressed as follows:

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j)$$

Where $2m$ is the total edge weights in the graph, A_{ij} represents the edge weight between nodes i and j, k_i and k_j are the sum of edge weights for node i and j respectively, c_i and c_j are the communities of the nodes, and δ is an indicator function.

We use the following formula to calculate the modularity change of moving node i to community C:

$$\triangle Q(i \to C) = \left[\frac{\sum_{in} + k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m} \right)^2 \right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m} \right)^2 - \left(\frac{k_i}{2m} \right)^2 \right] = \frac{k_{i,in}}{2m} - \frac{k_i \sum_{tot}}{2m^2}$$

Where $k_{i,in}$ is the total weights of edges between node i and community C, $\sum_{in}$ is the total weights of self-edges of community C, and $\sum_{tot}$ is the total weights of all edges of community C.

Similarly, we can derive the modularity change of moving node i out of community D as:

$$\triangle Q(D \to i) = -\left(\frac{k_{i,in}}{2m} - \frac{k_i \left(\sum_{tot} - k_i \right)}{2m^2} \right)$$

We used the above formula to calculate modularity change when implementing Louvain's algorithm.

4.3 Graph embedding

4.3.1 Node2Vec

Another method for community detection is graph embedding. The basic idea is to embed nodes so that the distances in the embedding space reflect node similarities in the original network. There are many approaches for measuring node similarity, and in this work we use random walk approach by node2vec[5]. We apply node2vec algorithm on the Steam graph with $p = 1$ and $q = 0.5, 1, 2$. We use the default settings for other parameters, so each node will be represented as a 128-d embedding vector. We separate the game nodes from user nodes, and assign communities according to the community detection results by modularity optimization. We visualize the results using t-SNE. Figure 2 shows the community clusters for game nodes (upper panels) and user nodes (bottom panels) with 2-d projection of node embedding by t-SNE.

In node2vec algorithm, different ratios of p/q give us flexibility to perform different tasks. A small q performs DFS, enables a macroscopic view, and is suitable for community detection, while a large q performs BFS, and gives us a structural view. Consistently, $q = 0.5$ and $q = 1$ performs better in community visualization especially for the User graph.

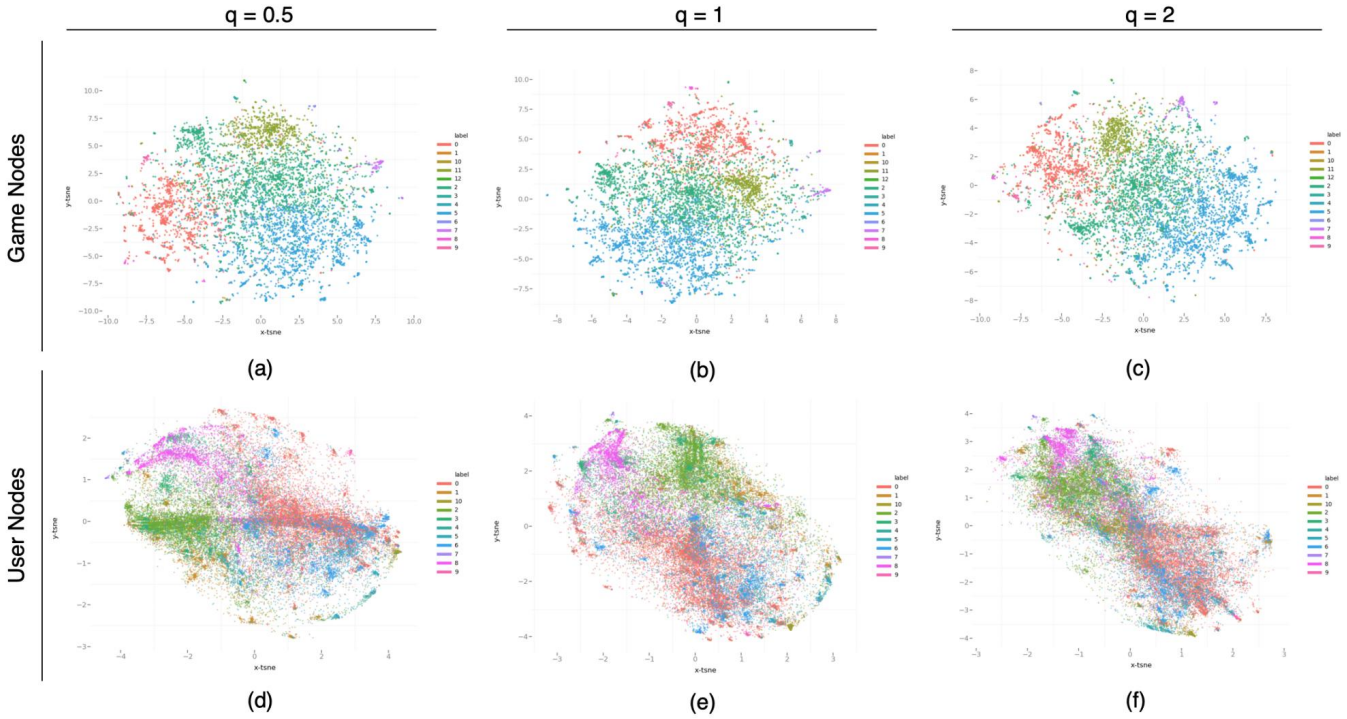

Figure 2: t-SNE plot of game/user node embeddings + community assignment

4.3.2 K-Means

During the Louvain community detection, we found that the size of some user communities is significantly larger than others, ranging from 300 to 12,000. This could be possible that the largest community represents the general players who are only playing the most popular games, and thus all behave similarly. In our case, since we want to choose user groups from clusters where the users are most close to each other, and also to reduce computational cost, we would like to further break down the large communities into smaller ones. Therefore, we apply k-means clustering separately on these larger communities (with size > 1000) from the louvain output with node2vec $q = 0.5$.

For each community, we want to find an optimal k value to break it down into k clusters. We experiment with different k values and use the silhouette score to evaluate the clustering results. The silhouette score is calculated using mean intra-cluster distance a and the mean nearest-cluster distance b for each sample.

$$s(i) = \frac{b(i) - a(i)}{max(a(i), b(i))}$$

We then find the optimal k value by comparing the average silhoette scores. As in Figure 3a, each community has a different optimal k value with the highest silhouette score. We plot the relation between the optimal k value vs the community size in Figure 3b, and found that generally they follow a proportional relation. This is reasonable since for larger communities, we may want to divide it into more clusters. With this optimal k-means cluster label assignment, we are able to break down the larger communities. Figure 3c visualizes an example k-mean clustering on the largest community (p = 1, q = 0.5) when k = 10.

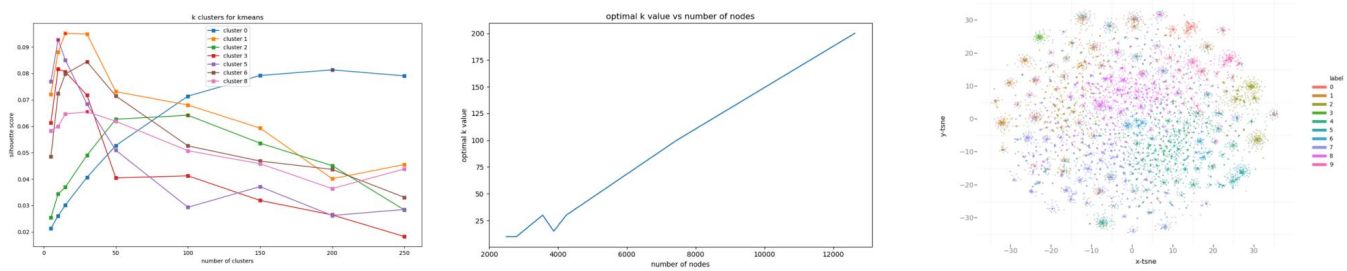

Figure 3: Kmeans clusters on the largest user community embedding

5 Personalized Bundle Recommendation to a Group of Users

5.1 Problem Formulation

While in many cases recommendation systems focus on single items, recent research has been conducted on the possibility of recommending a bundle of items. Additionally, sometimes it is in a party's best interest to consider recommendations for a group of users. This part of the project combines these two concepts by generating a bundle recommendation for a group of users based on video game data.

As it stands, there are already preexisting bundles of video games from the Steam dataset from which recommendations can be made to groups. However, we aim to personalize the bundles we end up recommending in order to see if we can better cater to users' tastes. With that being said, the overall challenge will be to make recommendations such that a group of users will prefer the personalized bundle to the existing bundles. If we achieve this, then we can count our recommendation system successful. We are also interested in how the type of group affects the quality of the recommendation, namely the randomness and the size of the group. Thus, we will compare recommendations for the user communities determined in section 4 of this paper to random groups of users across different group sizes.

5.1.1 Method Summary

The overall process of generating a personalized bundle recommendation for a group of users is as follows:

1. Train a model based on the user-game network that predicts a user's preference towards a game. We call this the $itemBPR$ model.

2. Learn parameters from $itemBPR$ model to then train a model that predicts a user's preference towards a bundle. We call this the $bundleBPR$ model.

3. Form a group of users for which to generate a bundle recommendation.

4. Using a greedy algorithm, generate a bundle recommendation for this group of users by averaging each user's bundle preference scores as predicted by the $bundleBPR$ model. Recommend the bundle with the highest average preference score.

5.1.2 Data Constraints

For the $itemBPR$ model, we create a User-Game network similar to the one used in section 4.1 such that its structure is also bipartite with edges between user nodes and game nodes indicating that

a user has positive sentiment towards (i.e. has purchased) a game; however, we will only consider games that already exist in bundles. We do this because, as stated in our Data Summary (section 3), bundles already contain the more popular games and account for around 70% of all item purchases. Moreover, we evaluate our recommender against existing bundles so keeping the selection of games consistent allows for fair comparison. For the *bundleBPR* model, we create a User-Bundle network, much like the bipartite User-Game network except that game nodes are replaced by bundle nodes from which an edge connecting a bundle node to a user means that the user has positive sentiment towards that bundle.

The notation we use to represent our data is as follows: let $U = u_1, ..., u_{|U|}$ be the set of users who have purchased at least one bundle, $I = i_1, ..., i_{|I|}$ the set of items (games) that exist in bundles and $B = b_1, ..., b_{|B|}$ the set of bundles, such that each $b_i \subseteq I$. For our personalized bundle recommendation system, $|U| = 29634$; $|I| = 2819$; and $|B| = 615$.

5.2 Item and Bundle Ranking Models

In order to determine a user's preference towards an item (a game) and a bundle (a group of games), we train models that will take into account a user's tastes to produce a personalized result. These models are based on the foundations of Bayesian Personalized Ranking (BPR), which were derived by Rendle et al in their paper *BPR: Bayesian Personalized Ranking from Implicit Feedback.*[10] The goal of BPR is to derive a personalized ranking $>_u$ over items (or bundles). To model the ranking, we assume an estimator $\hat{x} : U \times I \to \mathbb{R}$ that encodes the compatibility between a user $u \in U$ and an item $i \in I$, which is used to define the ranking:

$$i_p >_u i_n \leftrightarrow \hat{x}_{u,i_p} >_{\mathbb{R}} \hat{x}_{u,i_n}.$$

where i_p represents a positive item, i_n represents a negative item, $\hat{x}_{u,i_p}$ represents the estimate for compatibility between the user and a positive item, and $\hat{x}_{u,i_n}$ represents the same but for a negative item.

The first model we implement is an *itemBPR* model, accounting for a user's preference towards single items, and the second model is a *bundleBPR* model that applies information regarding item preferences (taken from the *itemBPR* model) to estimate user favor towards a bundle. Both models are trained with respect to an optimization, *BPROpt*. To determine *BPTOpt*, we use the following equation

$$BPROpt(\theta) = \ln p(>_u |\theta) p(\theta) = \sum_{(u,i_p,i_n) \in D} log\left(\sigma(\hat{x}_{u,i_p}(\theta) - \hat{x}_{u,i_n}(\theta))\right) - \lambda \parallel (\theta) \parallel^2$$

where, σ is the sigmoid function, θ is the parameter vector of the compatibility function, D represents the training set, λ is the regularization hyper-parameter, $\hat{x}_{u,i_p}$ and $\hat{x}_{u,i_n}$ represent compatibility estimates that the user u would purchase item p and n respectively. After deriving *BPROpt*, we maximize using a gradient descent based algorithm, specifically stochastic gradient descent with bootstrap sampling. In their paper, Rendle et al. demonstrate that stochastic gradient descent that chooses uniformly distributed user-item triples randomly with each update performs more efficiently than traversing the data user-wise with each update.[10]

5.2.1 Item BPR Model

This model aims to estimate a user's preference towards an item. The training data used in this model is a list of triplets (u, i_p, i_n) where u represents a user, i_p denotes a positive item (purchased by the user), and i_n denotes a negative item (not purchased by the user). The test/train sets are

divided with an 80/20 ratio. In this model, the estimator function used to calculate user preference towards an item, $\hat{x}_{u,i}$, is based on matrix factorization, and corresponds to the following equation

$$\hat{x}_{u,i} = \beta_i + P_i \cdot Q_i$$

where β_i is an item parameter, P_u is a k-dimensional latent parameter vector for a user, and Q_i is a k-dimensional latent parameter vector for an item.

5.2.2 Bundle BPR Model

This model aims to calculate a user's preference towards a bundle. The training data used in this model is a list of triplets (u, b_p, b_n) where u represents a user, b_p denotes a positive bundle (purchased by the user), and b_n denotes a negative bundle (not purchased by the user). Similar to in the *itemBPR* model, the test/train sets for the *bundleBPR* model are divided with an 80/20 ratio. In this model, the estimator function used to calculate user preference towards a bundle, $\hat{x}_{u,b}$, makes use of the parameters β, P, and Q that were learned from the *itemBPR* model, in the following equation

$$\hat{x}_{u,b} = \frac{1}{B_b} \sum_{i \in B_b} [\kappa\beta_i + (\mu P_i) \cdot (\omega Q_i] + C c_b + N_b$$

where μ and ω are k x k dimensional matrix adjustment parameters for P and Q. c_b represents the mean pair-wise Pearson correlation of the items in a bundle b, and N is $max_{j \in B}|B_j|$ that moderates bundle sizes by rewarding or penalizing them in order to prevent generating bundles of arbitrarily large sizes.

5.2.3 Evaluation of Models

Both the *itemBPR* and *bundleBPR* models are evaluated by computing the AUC metric. This calculates the fraction of times the model in question ranks positive items higher than negative items correctly when applied to the test set. Formally defined,

$$AUC = \frac{1}{T} \sum_{(u,p,n) \in T} \delta(\hat{x}_{u,p} - \hat{x}_{u,i} > 0)$$

where δ is the indicator function and T is the fraction of the data withheld for testing.

The models we trained for this paper produced an AUC of 0.84688 for *itemBPR* and 0.89798 for *bundleBPR*. Since both models score highly in ranking true positive items higher than true negative items for users in our test sets, we are confident in using them to determine rankings that will result in bundle recommendation for a group of users.

5.3 Bundle Generation and Recommendation

5.3.1 Bundle Generation Algorithm

We adopt a greedy algorithm to generate a personalized bundle of games that best serves as a recommendation to a group of users, G. This algorithm utilizes preference scores calculated by our *bundleBPR* model to guide user tastes as new bundles are presented. For a group of users, we take the average of preference scores calculated by our *bundleBPR* model for each user in the group as the group's preference score, and the algorithm updates based on this group rating. Taking the average is a fairly common way of assessing group preference when no explicit group information is available.[8]

The algorithm's steps are as follows

1. Create an initial bundle b that contains S randomly-chosen items
2. Randomly select k items $I_* \subset I \setminus \{i \in b\}$
3. Form a set B_* of new bundles that consist of additions or substitutions to b using items from I_*, or removing items from b
4. Through our *bundleBPR* model, determine preference scores of every user $u \in G$ for bundles $b' \in B_*$ relative to the initial bundle b i.e. $\hat{x}_{u,b'} - \hat{x}_{u,b}$
5. Let b_* be the bundle that has the highest average preference score P among the users in G, i.e. $b_* = argmax_{b' \in B_*}\,(avg_{u \in G}(\hat{x}_{u,b'} - \hat{x}_{u,b}))$ and $P = max_{b' \in B_*}(avg_{u \in G}(\hat{x}_{u,b'} - \hat{x}_{u,b}))$. If $P > 0$, accept b_* as the new bundle; else, accept b_* with diminishing probability
6. Repeat steps 2-5 until convergence where the final bundle that is returned by the algorithm serves as the recommendation

5.3.2 Evaluation Method

To evaluate the quality of our bundle recommendations to groups of users, we will assess if a group prefers existing bundles in the Steam dataset to a newly generated, personalized bundle. Using the *bundleBPR* model, we will calculate the group's preference score towards a bundle by averaging individual user preferences towards the bundle. The average rank R of a recommended bundle b_* is the number of times a group G prefers an existing bundle $b \in B$

$$R = |\{b \in B | avg_{u \in G}(\hat{x}_{u,b} - \hat{x}_{u,b_*}) > 0\}|$$

Intuitively, a personalized bundle b_* is a successful recommendation if it maintains a low average rank since that means we have improved upon the existing bundles by providing a group with a new bundle its users prefer more. Quite literally, an average rank of 0 means that there are no current bundles in the dataset that a group prefers more than the recommendation. Since the goal of this recommendation system is to provide a more personalized approach to bundle generation, beating any existing product is ideal.

5.3.3 Experiments and Results

For the following experiments, we implement the bundle generation algorithm outlined in section 5.3.1 using $S = 3$ and $k = 5$, where S is the size of the initial bundle and k is the number of random candidate items chosen in each iteration. We first perform community detection on the User graph, from which we divided the user nodes into 11 communities. We randomly draw groups out of each community for bundle generation, and we also draw random groups from the entire dataset as a control. We compare the performance of the algorithm by average bundle rank. The lower the rank is, the better recommendation it provides.

We randomly pick 1, 3, 10, 30 and 100 users from our communities 10 times and aggregate the resulting groups. We do the same random selection – this time from all possible users in the dataset – to form 10 aggregate randomly selected groups. For each group size, we calculate the aggregate bundle size, average bundle size, and average rank, and plot against the user group size. Figures 4a and 4b show that the aggregated bundle size and average bundle size is not different with or without community assignment. However, Figure 4c shows that the average ranks of bundles generated for user groups within individual communities are lower than that from all users. To further validate our conclusion, we randomly pick 1 and 10 users from our communities, as well as users from the entire user dataset, 80 times for bundle generation. Consistently, we observed no

difference in average rank for single user bundle generation, but significantly better performance for user groups from same the community vs. all users. These results demonstrate that community detection and user clustering significantly improve group bundle recommendation.

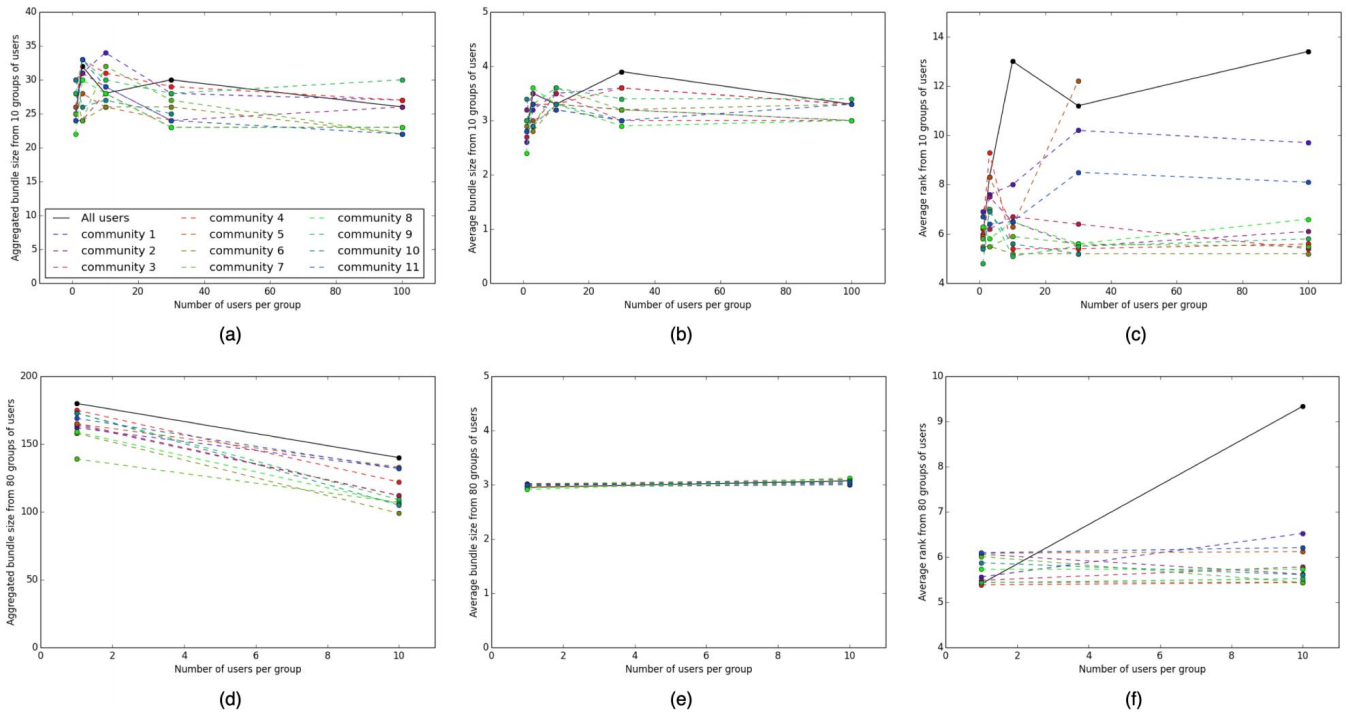

Figure 4: Bundle recommendation to groups of users made up from the entire graph and from within individual communities

6 Conclusion

This piece of research demonstrates how to effectively conduct group recommendations. We first consider the community structural properties of a dataset in order to determine how to best group users for recommendation. Following that, we apply the principles of Bayesian Personalized Ranking to a greedy algorithm, which generates a bundle of items according to group preference. Finally, we make recommendations based on that bundle generation and experiment on real-life data. Our results are clear: not only does the recommendation system perform significantly better when community information is considered in the formation of groups, but it also presents groups with bundles that are more preferred than those that currently exist in the system, demonstrating the desire for personalization.

However, the recommendation for user groups from the largest user community does not perform as well as that in other smaller communities. We have attempted to further divide the largest user community by kMean clustering after graph embedding. We did not observe significant improvement (data not shown). It is possible that this community does not have obvious clustering properties and can not be further divided into smaller parts. Future work may focus on other features of the graph via link prediction and neighborhood detection.

Github Repo

https://github.com/wxy1224/bundle_recommendation

References

[1] L. Baltrunas, T. Makcinskas, and F. Ricci. Group recommendations with rank aggregation and collaborative filtering. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 119–126, 2010.

[2] P. Bertens, A. Guitart, P. P. Chen, and Á. Periáñez. A machine-learning item recommendation system for video games. *arXiv preprint arXiv:1806.04900*, 2018.

[3] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[4] T. De Pessemier, S. Dooms, and L. Martens. Design and evaluation of a group recommender system. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 225–228, 2012.

[5] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[6] E. Ntoutsi, K. Stefanidis, K. Nørvåg, and H.-P. Kriegel. Fast group recommendations by applying user clustering. In P. Atzeni, D. Cheung, and S. Ram, editors, *Conceptual Modeling*, pages 126–140, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[7] A. Pathak, K. Gupta, and J. McAuley. Generating and personalizing bundle recommendations on steam. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1073–1076. ACM, 2017.

[8] S. Qi, N. Mamoulis, E. Pitoura, and P. Tsaparas. Recommending packages with validity constraints to groups of users. *Knowl. Inf. Syst.*, 54:345–374, 2018.

[9] M. Ramalingam. Design of quality-based recommender system for bundle purchases. 2013.

[10] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: bayesian personalized ranking from implicit feedback. *CoRR*, abs/1205.2618, 2012.

[11] T. Zhu, P. Harrington, J. Li, and L. Tang. Bundle recommendation in ecommerce. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, pages 657–666, 2014.

Contribution

Francesca Vera: Formulating problem; conducting related work research; summarizing Steam dataset; interpreting (mathematically) and explaining group bundle recommendation algorithm; analyzing recommendation results; designing poster.
Xiaoyan Wu: Constructing Steam graphs and generating node degree distribution plots; generating t-SNE visualizations with node2vec embeddings and Louvain cluster results; applying KMeans clustering algorithm to break down large user communities; writing up the report on relevant sections.
Yulian Zhou: Reconstructing and projecting the graph, and performing preliminary analysis and plotting degree distribution; coding up the Louvain algorithm for community detection; utilizing Node2Vec package for random walk and embedding; performing bundle generation with community information and plotting the graph; writing up the report on relevant section.