Outfit Generation and Recommendation: Efficient, Effective, and Adaptive Algorithms

Bryan Nathanael Wijaya
School of Electrical Engineering, KAIST
Daejeon, Republic of Korea
bryannwijaya@kaist.ac.kr

Quang Minh Nguyen
School of Electrical Engineering, KAIST
Daejeon, Republic of Korea
qm.nguyen@kaist.ac.kr

ABSTRACT

Fashion outfit recommendation is an essential component of online shopping, helping users access suitable outfits conveniently and identifying fashion items that appropriately match each other. However, sparse and implicit feedback as well as a lack of multimodal data (e.g., text and image description of items) pose a challenge to both conventional (e.g., matrix factorization) and modern (e.g., neural collaborative filtering and graph learning) methods. To this end, we propose *efficient, effective, and adaptive* algorithms, utilizing simple frequent itemset and graph aggregation properties. In particular, we consider two tasks:

Task 1 **Outfit Recommendation.** Given a dataset of user-itemset interactions (clicks users made on fashion outfits), classify unseen interactions as either true or false.

Task 2 **Outfit Generation.** Given a dataset of items in each itemsets, predict a missing item in incomplete itemsets.

For task 1, we propose SEGMENTED FREQUENT ITEMSETS, an algorithm which examines frequent itemsets among different user communities. For task 2, we propose NEIGHBOR MINING, an approach which takes into account the occurrences of neighboring items to produce rankings. Our theoretical analysis and experiments show that the proposed algorithms achieves reasonable performance within reasonable allocated compute budget while allowing for potential adaptation to different domains. Specifically, SEGMENTED FREQUENT ITEMSETS achieves an accuracy of **0.7374** while NEIGHBOUR MINING results in an average accuracy of **0.2485** and average rank of **82.562**.

Keywords: implicit feedback, collaborative filtering, recommender system, frequent itemset mining, graph mining

1 INTRODUCTION

The problem addressed in this project can be divided into two tasks, with the data, output, and objective as outlined below.

GIVEN datasets on user-item interaction, user-outfit itemset interaction (TR, VQ, VT, TQ), and outfit itemset-item affiliation (TR, VQ, VT, TQ) [1]

FIND (1) Outfit Recommendation: a boolean on whether a given outfit itemset correctly satisfies the fashion taste of a given user and (2) Outfit Generation: the top 100 item candidates for the missing item in an unseen outfit itemset

GOAL (1) Outfit Recommendation: maximize the accuracy of prediction and (2) Outfit Generation: maximize the accuracy and minimize the ranking of candidates.

[1]TR: training, VQ: validation query, VT: validation ground truth, TQ: test query.

The problem of interest in this project has a great potential impact on the online fashion community: outfit recommendation algorithms will allow users to conveniently find and purchase the fashion outfits that fit their taste from the many possible choices; for fashion companies, efficient outfit generation algorithms will result in more joint purchases.

However, with sparse and implicit feedback as well as a lack of external, multimodal data features (e.g., text and image description of items and itemsets), both conventional (e.g., matrix factorization) and modern (e.g., neural collaborative filtering and graph learning) methods fail to provide accurate predictions within practical and limited compute budget. We argue that the underlying problem lies in limited *latent* structures in the problem at hand, which the said methods try to capture through user and item embeddings. As such, we design novel algorithms which instead utilise *explicit* structures in the form of *frequent itemsets* and *items*. Accordingly, our algorithmic contribution is two-fold:

(1) We introduce SEGMENTED FREQUENT ITEMSETS, an algorithm which segments the user population into communities and find frequent itemsets within each community for the task of **outfit recommendation**
(2) We introduce NEIGHBOR MINING, an algorithm which aggregates neighborhood information of items in an itemsets for the task of **outfit generation**.

To evaluate and examine properties of SEGMENTED FREQUENT ITEMSETS and NEIGHBOR MINING, we perform brief theoretical analyses and conduct extensive experiments with various hyperparameter settings and compare them to conventional baselines. Our results suggest that the proposed algorithms provide reasonable performance, require low compute resources, and allow for potential adaptation to different domains.

The rest of this report is organised as follows: in Section 2, we briefly describe related work in implicit feedback collaborative filtering and frequent itemset mining; Section 3 provides details regarding our proposed algorithms as well as baselines; we present empirical results in Section 4; finally, Section 5 serves to summarize our findings and suggest future directions.

2 RELATED WORK

Our work is related to two lines of research.

2.1 Implicit feedback collaborative filtering

Collaborative filtering recommender systems aim to infer users' preferences for items from preferences of other users [1]. Different from traditional settings, e.g., movie rating, where preferences are collected as a wide range of numerical values, implicit feedback datasets only consider binary interactions between users and items [2], which aligns with our problem. A common and simple strategy is matrix factorization, which models interactions as dot products between vectors embedding users and items [3]. More modern methods also assume user and item embeddings while relying on multilayer perceptrons, i.e., neural collaborative filtering [4], or graph neural networks, i.e., neural graph collaborative filtering [5], to learn accurate interaction models. However, recent work has pointed out that such modern methods are highly inefficient while offering limited improvement over simpler approaches [6]; furthermore, reported state-of-the-art results are also criticized as not replicable, especially for implicit feedback [7]. Our experiments verify this drawback on not just neural collaborative filtering models but also matrix factorization.

2.2 Frequent Itemset Mining

The goal of frequent itemset mining is to find regularities in collections of items [8]. This work employs simple ideas from frequent itemset mining to find itemsets that are the most likely to be interacted with and items that are the most likely to be included in itemsets. Our results demonstrate that frequent itemsets mining can extract global features that benefit outfit recommendation and outfit generation.

3 METHODOLOGY

3.1 Task 1: Outfit Recommendation

We provide a detailed description of our proposed algorithm as well as baselines and competitors. The common notation we will use are n_{user} for the number of users, $n_{itemset}$ for the number of itemsets, and $n_{interaction}$ for the number of samples in the training data.

3.1.1 Segmented Frequent Itemsets (SFI). Our proposed algorithm, Segmented Frequent Itemsets, groups users based on a clustering/community detection of choice on the bipartite user-itemset graph, then classifies a given user-itemset interaction as true if the itemset is among the $n_{fi} \in \mathbb{N}$ most frequent itemsets within the cluster of the user (refer to Figure 1). The hyperparameters in this algorithm include

(1) Clustering algorithm: there are many choices, such as Louvain algorithm [9] or Fluid Community algorithm [10] on the derived unipartite graph between users. In our experiment, we use singular value decomposition to obtain user and item embeddings of dimension n_embed, which are then clustered using k-means algorithm.
   - $n_{embed} \in \mathbb{N}$: the number of embedding dimensions is a hyperparameter that results from our specific choice of clustering algorithm, but otherwise is not necessary.
   - $n_{cluster} \in \mathbb{N}$: the number of clusters

(2) $n_{fi} \in \mathbb{N}$: the optimal value of this hyperparameter might be highly dependent on the domain; we experiment with multiple different values in Section 4.

SFI is theoretically guaranteed to be efficient:

(1) Truncated singular value decomposition on a sparse matrix can be computed in negligible time, using efficient eigensolvers such as ARPACK [11].

(2) Finding $n_{cluster}$ clusters of users using k-means algorithm for n_{iter} iterations takes $O(n_{iter} n_{cluster} n_{user} n_{embed})$ time. This complexity is dominated only by n_{user}.

(3) With precomputed clustering, the complexity of generating itemset counts and sorting them is $O(n_{interaction} + n_{itemset} \log n_{itemset})$.

3.1.2 Baseline 1: Random Guessing (RG). For this baseline, we randomly classify a user-itemset interaction as true or false with equal probability.

3.1.3 Baseline 2: Frequent Itemsets (FI). For this baseline, we classify a user-itemset interaction as true if the itemset is one of the 1000 most frequent itemsets in the entire user-itemset interaction data.

3.1.4 Baseline 3: Shortest Path Length (SP). For this baseline, we classify a user-itemset interaction as true if the shortest path length between them in the given bipartite interaction graph is 3.

3.1.5 Competitor 1: Matrix Factorization (MF). For this competitor, we randomly initialise learnable user embeddings $U \in \mathbb{R}^{n_{user} \times n_{embed}}$ and item embeddings $V \in \mathbb{R}^{n_{itemset} \times n_{embed}}$, as well as user biases $b_U \in \mathbb{R}^{n_{user}}$, itemset biases, $b_V \in \mathbb{R}^{n_{itemset}}$, and a common bias $b \in \mathbb{R}$. Furthermore, we produce $n_{negative}$ negative samples, i.e., interactions not present in the training dataset, for each true interaction (also referred to as positive samples). Then the weighted mean squared error

$$L(U, V, b_U, b_V, b) = \mathbb{E}_{(i,j)}[w_{i,j}(\hat{y}(i,j) - y(i,j))^2]$$

is minimised, where

$$\hat{y}(i,j) = \langle u_i, v_j \rangle + b_{U,i} + b_{V,j} + b$$

is the predicted interaction, $y(i,j) \in \{0,1\}$ is the ground-truth interaction, $w_{i,j}$ is a positive weight assigned to the interaction (i,j) depending on whether it is a negative or positive sample, and the empirical expectation is over our interaction dataset together with negative samples. Some important hyperparameters for this competitor include:

(1) $n_{embed} \in \mathbb{N}$
(2) $n_{negative}$
(3) $w_{i,j} \in \mathbb{R}$: in our experiment, this weight is lower for negative samples and higher for positive samples.

3.1.6 Competitor 2: Neural Collaborative Filtering (NCF). For this competitor, we also randomly initialise learnable user embeddings $U \in \mathbb{R}^{n_{user} \times n_{embed}}$ and itemset embeddings $V \in \mathbb{R}^{n_{itemset} \times n_{embed}}$. For each user-itemset interaction, we concatenate their embeddings and input the combined vector to a multilayer perceptron, which outputs a prediction $\hat{y} \in \mathbb{R}$. We also use the weighted mean squared error as objective just as in matrix factorization. Some important hyperparameters include:

(1) $n_{embed} \in \mathbb{N}$
(2) $w_{i,j} \in \mathbb{R}$: in our experiment, this weight is lower for negative samples and higher for positive samples.

We also consider the following methods, used as part of our ablation study:

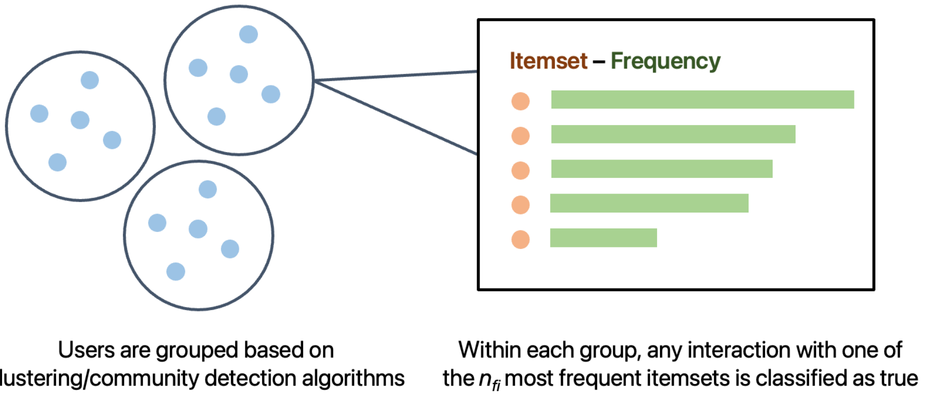

Itemset – Frequency
Users are grouped based on clustering/community detection algorithms
Within each group, any interaction with one of the n_{fi} most frequent itemsets is classified as true

Figure 1: Our proposed algorithm for Task 1, Segmented Frequent Itemsets

3.1.7 Ablation 1: Adaptive Frequent Itemsets (AFI). For this baseline, we classify a user-itemset interaction as true if the itemset is one of the n_{fi} most frequent itemsets in the entire user-itemset interaction data. This is a generalization of the baseline FI and omits the clustering process from SFI. The only hyperparameter is $n_{fi} \in \mathbb{N}$.

3.1.8 Ablation 2: Singular value decomposition (SVD). For this competitor, we find the truncated SVD of the binary user-itemset interaction matrix, resulting in user embeddings $U \in \mathbb{R}^{n_{user} \times n_{embed}}$ and itemset embeddings $V \in \mathbb{R}^{n_{itemset} \times n_{embed}}$. A user-itemset interaction (i, j) is then classified as true if $\langle u_i, v_j \rangle > 0$. This competitor uses singular value decomposition just as our considered version of SFI but omits clustering and frequent itemset mining. The only hyperparameter is $n_{embed} \in \mathbb{N}$.

3.2 Task 2: Outfit Generation

We provide two baseline algorithms and propose three prospective algorithms for the task of Outfit Generation as described in the following. The second approach of Neighbor Mining is used as the final model for this task after evaluation.

3.2.1 Baseline: Random guessing (RG). In this algorithm, for each queried itemset, we take 100 random items that occur in the itemset-item training dataset and return this list to the query.

3.2.2 Baseline: Frequent items (FI). For this algorithm, we analyzed the itemset-item affiliation training dataset and sort the items that appear in the dataset in the descending order of their occurrence. In the **first approach**, the first 100 most frequent items are recommended to all of the queries. In the **second approach**, the first 100 most frequent items that are not readily included in the unseen itemset of interest are recommended to each query.

3.2.3 Extended itemset-itemset collaborative filtering (eISCF). There are three approaches investigated for this part. In the **first approach**, for each row of the query, we scanned through the itemset-item affiliation training dataset and sort all the itemsets in the training dataset in the descending order of their similarity to the queried itemset. The similarity of two itemsets A and B is calculated as in (1) because the itemset-item matrix is sparse. In particular, the minimum value of similarity between two itemsets is α, which occurs when the two itemsets do not have any item in common.

$$sim(A, B) = n(A \cap B) + \alpha \tag{1}$$

Next, for each of the itemsets in the training dataset, their corresponding items are accumulated to the list of candidates together with the similarity of that itemset with respect to the queried itemset, if the item is not already in the queried itemset. The candidates are then sorted in a descending order of their accumulated similarity value and the first 100 candidates are returned to the query.

The **second approach** is similar to the first one, but now the similarity is calculated as (1) with $\alpha = 0$ and itemsets with zero similarity (i.e., no common items with the queried itemset) are not included in the list of sorted itemsets. Then, for each of the itemsets with positive similarity value, their corresponding items are accumulated to the list of candidates together with the similarity of that itemset with respect to the queried itemset, if the item is not already in the queried itemset. Next, the candidates are sorted in a descending order of their accumulated similarity value and frequent items (in the descending order of their frequency in the training dataset) that are not already in the list nor the queried itemset are added to the list of candidates to give a total of exactly 100 candidates, which is then returned to the query.

The **third approach** resembles that of the first approach, but it now additionally utilizes the user-item interaction dataset too, by the assumption that a user, which has a particular fashion taste, can be thought of approximately as an itemset. Here, since a user C is only hypothetically an approximate itemset, we manipulated its contribution to the accumulated similarity by imposing a penalty to its similarity with respect to the queried itemset A as shown in (2). The similarity between two itemsets A and B in (1) is adjusted accordingly based on the penalty β as in (3). The value of α in this approach is fixed to the value that was found optimal in the first approach.

$$sim(A, C) = (1 - \beta)(n(A \cap C) + \alpha) \tag{2}$$

$$sim(A, B) = \beta(n(A \cap B) + \alpha) \tag{3}$$

Now, for each query itemset, after accumulating the items and their corresponding parent itemset (or user) similarity value (excluding items that are already in the queried itemset) for all itemsets and users in the training dataset and user-item dataset, the candidates are then sorted in a descending order of their accumulated similarity value and the first 100 candidates are returned to the query.

3.2.4 Neighbor Mining (NM).

In this novel method, there are two approaches considered. In the **first approach**, we accumulated the neighboring items and their number of neighboring occurrences for each item by traversing through the training dataset. Here, two items are said to be neighboring n times if they appear together in n itemsets. Then, for each queried itemset, we pass through each of its items and accumulate the neighbors of the item and the corresponding neighboring occurrence to the list of candidates, based on the learning result from the training dataset. Finally, we remove the items in the list of candidates if they are already included in the queried itemset, sort the candidates in the descending order of neighboring occurrence, and add frequent items (in the descending order of their frequency in the training dataset) that are not already in the list nor queried itemset to give a final list of 100 candidates, which is then returned to the query.

In the **second approach**, we additionally incorporated the user-item interactions by the assumption that a user, which has a particular fashion taste, can be thought of approximately as an itemset. Again, since a user is only hypothetically an approximate itemset, we manipulated its contribution to the accumulated number of item-item neighboring occurrence by imposing a weight ratio γ to its neighboring occurrence number as shown in (4). In particular, if an item is neighboring with another item n times in the user-item dataset, its number of neighboring occurrence will be γn.

$$n(\text{neighboring})_{\text{U}-\text{I}} = \gamma n(\text{neighboring})_{\text{IS}-\text{I}} \tag{4}$$

Hence, here, we accumulated the neighboring items and their number of (adjusted) neighboring occurrences for each item by traversing through the training and user-item datasets. Then, for each queried itemset, we pass through each of its items and accumulate the neighbors of the item and the corresponding neighboring occurrence to the list of candidates, based on the learning result from the training and user-item datasets. Finally, we remove redundant items, sort, add frequent items accordingly, and return to the query exactly like the first approach. This approach of Neighbor Mining is used as the final model for Outfit Generation as it is significantly faster and gives the most optimum accuracy and rank compared to other algorithms, as shown later. Figure 2 shows a simplified illustration of the second approach of Neighbor Mining with $\gamma = 0.2$, where we assume to search for the top 3 item candidates (instead of 100).

3.2.5 Extended item-item collaborative filtering (eICF).

There are three approaches investigated for this part. For all of them, We begin by transposing the itemset-item training dataset and user-item affiliation dataset from "which items are affiliated to each itemset/user?" into "which itemsets/users are affiliated to each user?". In technical perspective, now the dictionary keys are the item numbers (formerly, the itemset/user number) and the value of each key is the list of itemsets/users affiliated to the item (formerly, the list of items affiliated to the itemset/user).

In the **first approach**, for each row of the query, we scanned through each item in the transposed training dataset, sorted the items in the query row in the descending order of their similarity to the item of concern in the transposed training dataset, summed the similarity of all the items in the query row with respect to the item of concern in the transposed training dataset, and save it as the weight of that training item of concern for that query row. The similarity between two items a and b is calculated as in (5), which is the number of itemsets they are both affiliated to in the transposed training dataset. The weight of a transposed training dataset item a for a queried itemset C is calculated as in (6).

$$sim(a, b) = n(a \cap b) \tag{5}$$

$$w(a) = \sum_{x \in C} sim(a \cap x) \tag{6}$$

After repeating this for all items in the training dataset, we have a list of (training item number, weight) tuples for that row of query. We then sort the predicted items for that queried itemset in the descending order of their weight value and remove items that are already included in the queried itemset. The first 100 candidates are then returned to the query.

The **second approach** is similar to the first one, but now an additional restriction is imposed. The similarity of an item in the queried itemset is summed into the weight of an item in the transposed training dataset for that queried itemset only if their similarity is at least a certain threshold σ. Hence, the weight of a transposed training dataset item a for a queried itemset C is now calculated as in (7). Apart from this, everything else is identical to the first approach.

$$w(a) = \sum_{x \in C, sim(a \cap x) \geq \sigma} sim(a \cap x) \tag{7}$$

The **third approach** resembles that of the first approach, but it now additionally utilizes the user-item interaction dataset too, by the assumption that a user, which has a particular fashion taste, can be thought of approximately as an itemset. Similarly to the training dataset, the user-item interaction dataset is also transposed. In this approach, for each row of the query, in addition of scanning through all the items in the transposed training dataset, we also scanned through all the items in the transposed item-user interaction dataset with the above-mentioned assumption. However, since a user is only hypothetically an approximate itemset, to determine the ratio of significance of the similarity obtained from the training dataset and the significance of the similarity values obtained from the user-item interaction dataset, we manipulated their contribution to the weight by a ratio hyperparameter δ as shown in (8). Note that the similarity between an item from the queried dataset and an item from the transposed item-user interaction dataset is calculated the same way like the similarity with respect to an item from the transposed training dataset as in (3), except that the intersection here occurs between the two items with respect to the transposed item-user interaction dataset, not the transposed training dataset. Apart from this, everything else is identical to the first approach.

$$w(a) = \sum_{x \in C} \delta sim_{\text{I}-\text{IS}}(a \cap x) + (1 - \delta) sim_{\text{I}-\text{U}}(a \cap x) \tag{8}$$

4 EXPERIMENTS

4.1 Task 1: Outfit Recommendation

Our experiments for this task serve to investigate three questions:

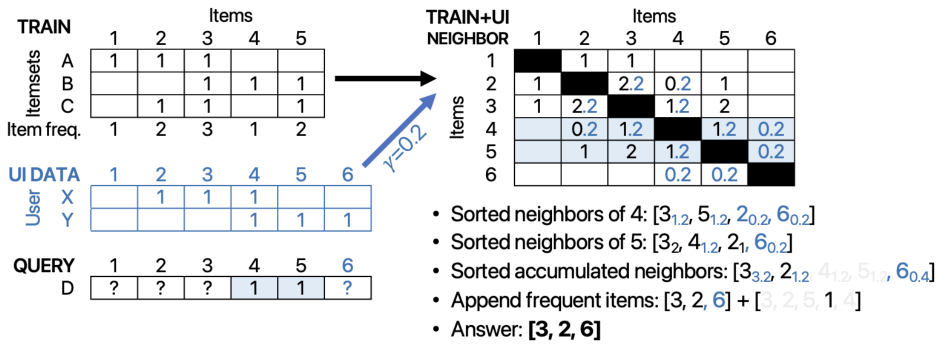

TRAIN
Items
Itemsets
Item freq.
UI DATA
User
QUERY
γ=0.2
TRAIN+UI NEIGHBOR
Items
• Sorted neighbors of 4: [3_{1.2}, 5_{1.2}, 2_{0.2}, 6_{0.2}]
• Sorted neighbors of 5: [3₂, 4_{1.2}, 2₁, 6_{0.2}]
• Sorted accumulated neighbors: [3_{3.2}, 2_{1.2}, 4_{1.2}, 5_{1.2}, 6_{0.4}]
• Append frequent items: [3, 2, 6] + [3, 2, 5, 1, 4]
• Answer: [3, 2, 6]

Figure 2: Neighbor Mining Approach 2 with $\gamma = 0.2$ to obtain the top 3 missing item candidates of a query. Texts in blue signify the difference from Approach 1 due to the influence of user-item dataset.

- **Q1: What are the effects of hyperparameters on the performance of SFI?** To answer this question, we conduct experiment **E1**, where the validation accuracy across $n_{embed} \in \{8, 16, 32\}$, $n_{cluster} \in \{10, 15, 20, 25, 30\}$, and $n_{fi} \in \{1000t\}_{t=1}^{7}$ is examined.
- **Q2: Does SFI yield higher accuracy than baselines and competitors?** To answer this question, we conduct experiment **E2**, where we compare the performance of SFI against that of the various baselines and competitors described above. Regarding matrix factorization and neural collaborative filtering, the considered hyperparameters are $n_{embed} \in \{16, 32\}$ and $w_{i,j} = 1$ for positive samples and $w_{i,j} = 0.2$ for negative samples. We also produce $n_{negative} = 3$ negative samples for each positive sample.
- **Q3: What are the contributions of different components of SFI, namely SVD embedding and frequent itemset mining, to the final performance?** To answer this question, we conduct experiment **E3**, an ablation study where the validation accuracy between SVD, AFI, and SFI are compared. We consider $n_{embed} \in \{8, 16, 32\}$ and $n_{fi} \in \{1000t\}_{t=1}^{7}$.

4.1.1 **E1**. The accuracy of SFI for varying parameters is plotted in Figure 3. Detailed results are included in Table 1. We observe that overall, as n_{fi} increase, the accuracy also increases until a certain point, i.e, $n_{fi} \in [5000, 6000]$ and then decreases. This agrees with intuition: the effect of popular itemsets being viewed only holds valid until a saturation point. Furthermore, lower embedding dimensions and higher number of clusters improve performance over their lower and higher counterparts, respectively.

4.1.2 **E2**. We compare between SFI and baselines as well as competitors in terms of validation accuracy in Table 2. SFI proves to be the best method with accuracy 0.7374. Especially for the case of NCF, we observe that the validation accuracy is kept constant at 0.5 during the learning process. Meanwhile, even though increasing the embedding dimension seems to improve the accuracy of MF, this method still yields poor results compared to baselines. These results can be viewed in Figure 6, where we plot the validation accuracy during the learning process, within our training budget of 2 epochs and 5 epochs for NCF and MF, respectively.

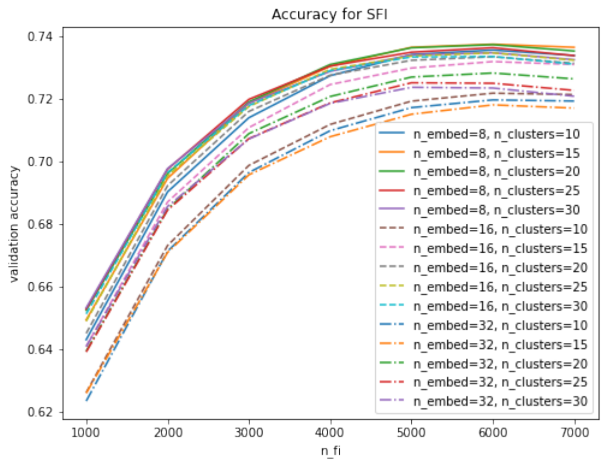


Figure 3: Experiment E1 for Task 1: How the performance of SFI changes with varying hyperparameters

.

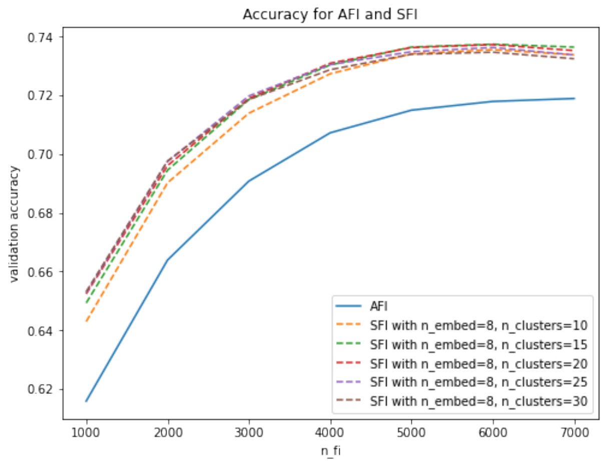


Figure 4: Experiment E3 for Task 1: Comparison between SFI and AFI

n_{embed}	$n_{cluster}$	n_{fi} = 1000	n_{fi} = 2000	n_{fi} = 3000	n_{fi} = 4000	n_{fi} = 5000	n_{fi} = 6000	n_{fi} = 7000
8	10	0.6429	0.6902	0.7139	0.7273	0.7341	0.7355	0.7338
8	15	0.6493	0.6945	0.7185	0.7303	0.7364	**<u>0.7374</u>**	0.7364
8	20	0.6523	0.6961	0.7190	0.7309	0.7363	0.7373	0.7352
8	25	0.6527	0.6974	0.7198	0.7304	0.7349	0.7363	0.7338
8	30	0.6531	0.6976	0.7186	0.7287	0.7340	0.7347	0.7325
16	10	0.6260	0.6730	0.6986	0.7118	0.7192	0.7218	0.7213
16	15	0.6408	0.6870	0.7107	0.7245	0.7298	0.7319	0.7309
16	20	0.6449	0.6922	0.7159	0.7275	0.7323	0.7334	0.7313
16	25	0.6489	0.6953	0.7175	0.7294	0.7336	0.7346	0.7323
16	30	0.6512	0.6964	0.7178	0.7290	0.7333	0.7335	0.7310
32	10	0.6233	0.6712	0.6964	0.7098	0.7171	0.7196	0.7192
32	15	0.6258	0.6708	0.6956	0.7078	0.7150	0.7180	0.7170
32	20	0.6393	0.6846	0.7088	0.7207	0.7269	0.7282	0.7263
32	25	0.6390	0.6844	0.7072	0.7187	0.7251	0.7249	0.7226
32	30	0.6407	0.6855	0.7071	0.7185	0.7236	0.7234	0.7208

Table 1: The validation accuracy of SFI for Task 1, with varying hyperparameters

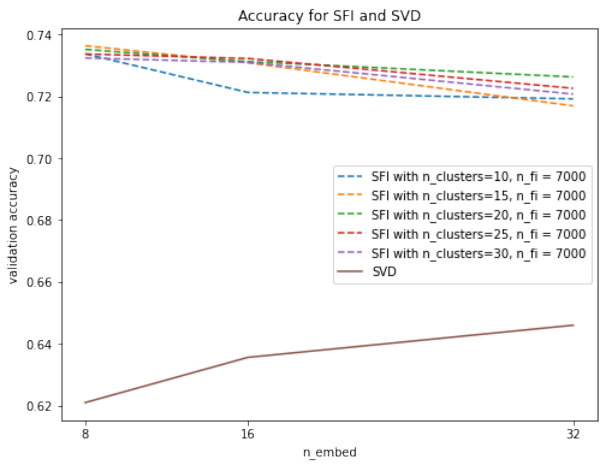


Figure 5: Experiment E3 for Task 1: Comparison between SFI and SVD

Method	Accuracy
SFI	**<u>0.7374</u>**
RG	0.5000
FI	0.6158
SP	0.6090
MF	0.5436
NCF	0.5008

Table 2: The performance of SFI compared to baselines and competitors

4.1.3 **E3**. We compare between AFI and SFI with varying n_{fi}; for SFI, we only use n_{embed} = 8 for simplicity. The result is plotted in Figure 4. SFI can be observed to consistently outperform AFI.

We also compare between SVD and SFI with varying n_{embed} while fixing n_{fi} = 7000. The result is plotted in Figure 5. Again, SFI consistently outperforms SVD, even with a further margin. As such, we can say that SFI efficiently combines SVD and AFI and inherits a large part of its predictive power from AFI.

4.2 Task 2: Outfit Generation

The performance of our algorithms in this task is determined by the accuracy and rank averaged over all the queries, which are calculated based on the querying result on the validation dataset. The accuracy and rank for each query row are calculated as in (9) and (10), respectively.

$$Accuracy = \begin{cases} 1, & \text{missing item} \in \text{candidates} \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

$$Rank = \begin{cases} \text{rank(ground-truth)}, & \text{missing item} \in \text{candidates} \\ 101, & \text{otherwise} \end{cases} \tag{10}$$

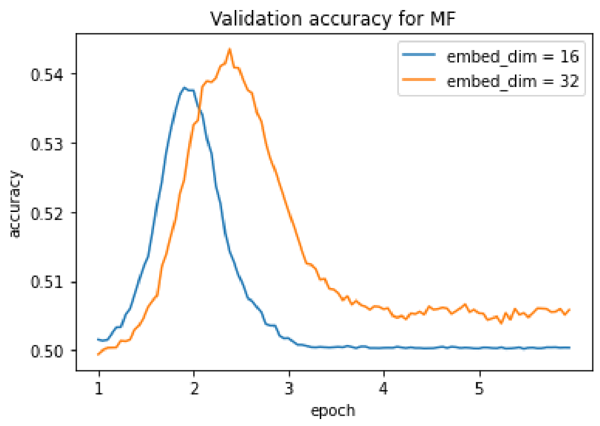
Table 3 tabulates the averaged accuracy and rank for each of the investigated methods that are obtained from hyperparameters that yield the highest accuracy and rank for each method.

Method	Accuracy	Rank	Remarks
RG	0.0014	100.953	N/A
FI	0.1170	93.388	Approach 2
eISCF	0.1827	87.799	Approach 1, α = 0.02
NM	**<u>0.2485</u>**	**<u>82.562</u>**	Approach 2, γ = 0.04
eICF	0.2477	82.616	Approach 3, δ = 0.96

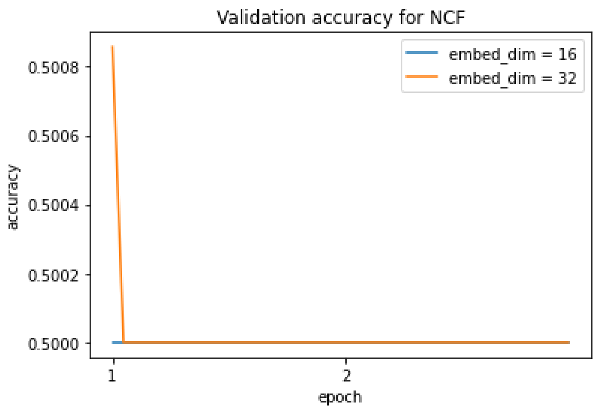
Table 3: Overview of Task 2 evaluation on validation dataset.

Table 4 tabulates the averaged accuracy and rank for each approach taken in the frequent itemsets (FI) method.

Table 5 tabulates the averaged accuracy and rank for each approach taken in the extended itemset-itemset collaborative filtering (eISCF) method, with varied hyperparameters. For the first and



(a) MF results in poor performance



(b) NCF cannot learn past random guessing

Figure 6: Results regarding MF and NCF in experiment E2 for task 1.

Approach	Accuracy	Rank
1	0.1166	93.410
2	**0.1170**	**93.388**

Table 4: Task 2 evaluation on validation dataset for FI. The approach with the best result is highlighted.

Approach	α	β	Accuracy	Rank
1	0.00	–	0.1362	90.101
1	0.02	–	**0.1827**	**87.799**
1	0.04	–	0.1575	89.615
1	0.06	–	0.1470	90.577
1	0.08	–	0.1437	91.057
1	0.10	–	0.1405	91.381
1	0.12	–	0.1376	91.654
1	0.14	–	0.1343	91.816
1	0.20	–	0.1318	92.146
1	0.30	–	0.1260	92.471
1	0.40	–	0.1253	92.624
1	0.50	–	0.1250	92.707
1	0.60	–	0.1231	92.813
1	0.70	–	0.1228	92.887
1	0.80	–	0.1228	92.931
1	0.90	–	0.1228	92.969
1	1.00	–	0.1224	92.983
2	0.00	–	**0.1665**	**88.744**
3	0.02	0.00	0.1065	93.909
3	0.02	0.20	0.1083	93.816
3	0.02	0.40	0.1091	93.677
3	0.02	0.06	0.1138	93.406
3	0.02	0.80	0.1293	92.399
3	0.02	1.00	**0.1827**	**87.799**

Table 5: Task 2 evaluation on validation dataset for eISCF. The best result for each approach is highlighted.

third approaches of eISCF with varying hyperparameters, refer to Figures 7–8 and Figure 9, respectively.

Table 6 tabulates the averaged accuracy and rank for each approach taken in the NEIGHBOR MINING (NM) method, with varied hyperparameters. For the second approach of NM with varying hyperparameters, refer to Figures 10–11.

Table 7 tabulates the averaged accuracy and rank for each approach taken in the extended item-item collaborative filtering (eICF) method, with varied hyperparameters. For the second and third approaches of eICF with varying hyperparameters, refer to Figure 12 and Figures 13–14, respectively.

Overall, we observe that the second approach of NM at $\gamma = 0.04$ achieves the highest accuracy and the best rank, so this approach is used to generate the prediction for the test queries of Task 2 (i.e., Outfit Generation, itemset_item_test_prediction.csv).

5 CONCLUSIONS

We proposed algorithms SEGMENTED FREQUENT ITEMSETS for task 1 (outfit recommendation) and NEIGHBOR MINING for task 2 (outfit generation), which are shown to be efficient, effective, and adaptive algorithms. Specifically, SEGMENTED FREQUENT ITEMSETS, utilizing user clustering and frequent itemset mining, outperforms all baselines and competitors, including even conventional and modern methods like learning-based matrix factorization or neural collaborative filtering; meanwhile, NEIGHBOR MINING, aggregating frequent neighbor information, outperforms collaborative filtering and other frequent item approaches. Through this project, we demonstrated that frequent itemset and frequent item approaches can be a promising direction for implicit feedback recommender systems and for itemset completion tasks, especially in the fashion domain that we investigated. We also confirm recent results that modern neural approaches are not necessarily effective in implicit feedback domains.

Our methods leave room for improvement. First, we have only investigated singular value decomposition embeddings and k-means

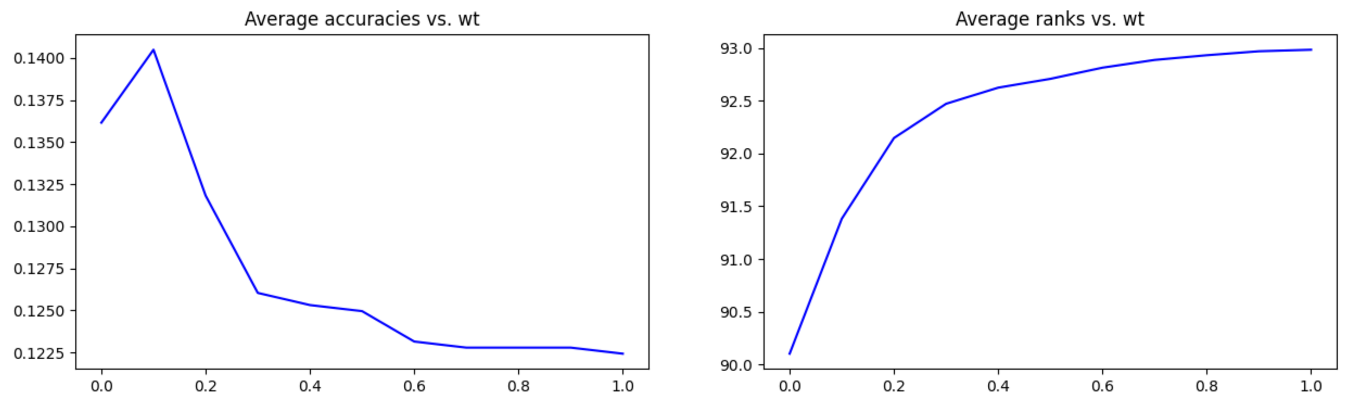


Figure 7: Accuracy and rank of the first approach of the eISCF method of Task 2 as α is varied from 0 to 1 with a step of 0.1. A local maximum in accuracy occurs at $\alpha \approx 0.1$.

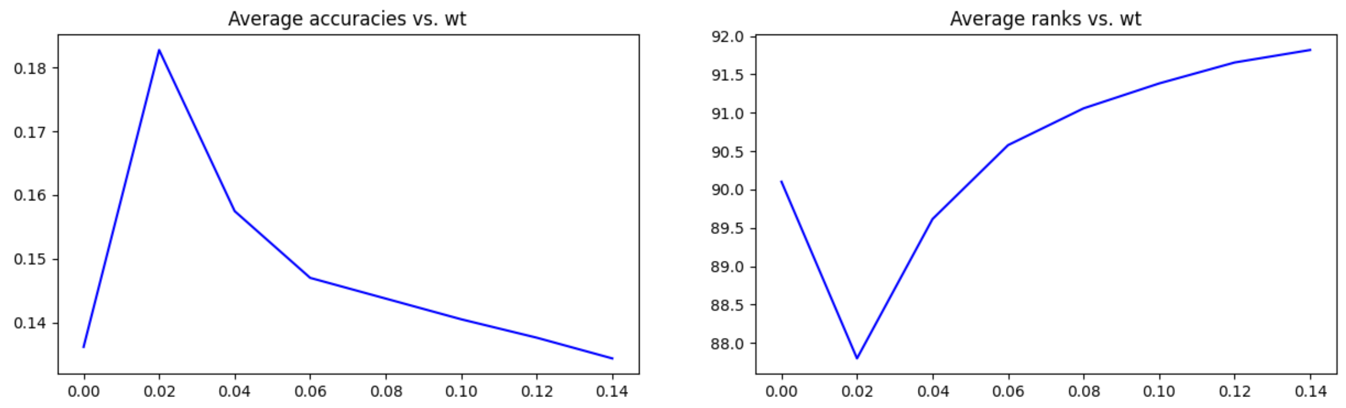


Figure 8: Accuracy and rank of the first approach of the eISCF method of Task 2 as α is varied from 0 to 0.14 with a step of 0.02. A local maximum in accuracy and a local minimum in rank both occur at $\alpha = 0.02$, which gives the best performance.

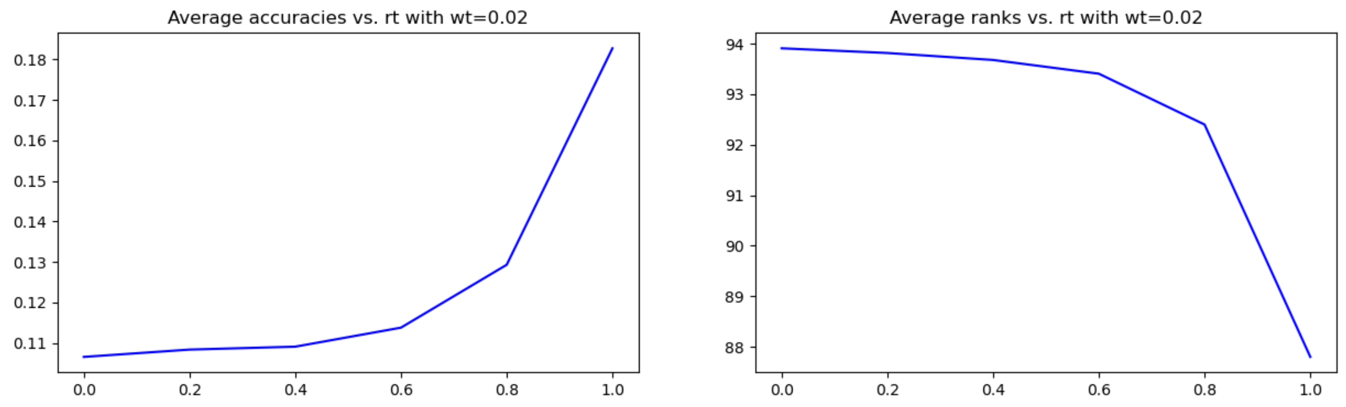


Figure 9: Accuracy and rank of the third approach of the eISCF method of Task 2 with $\alpha = 0.02$ as β is varied from 0 to 1 with a step of 0.2. The accuracy and rank both improved at an increased β penalty value, so it seems that incorporating user-item dataset in this method is not a good idea.

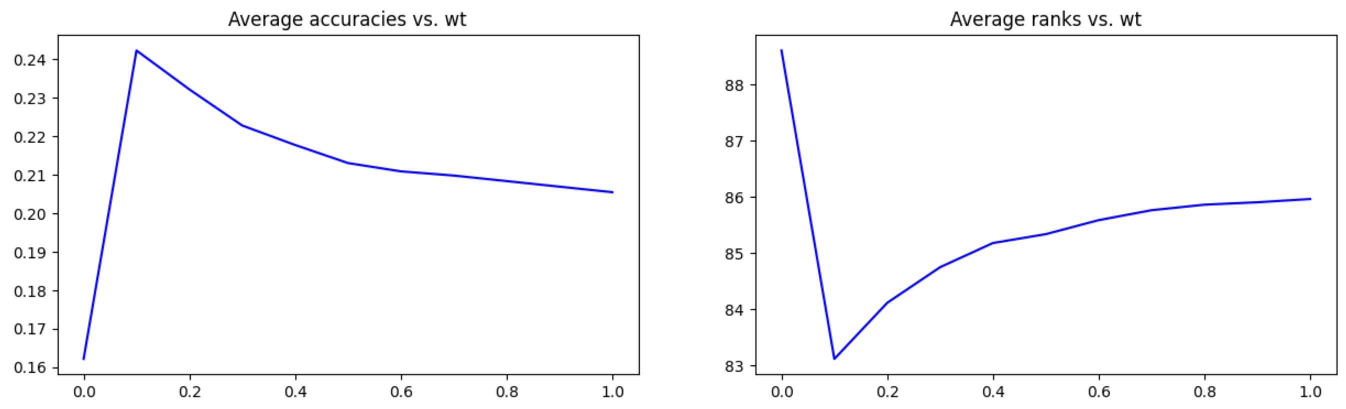


Figure 10: Accuracy and rank of the second approach of the NM method of Task 2 as γ is varied from 0 to 1 with a step of 0.1. A local maximum in accuracy and a local minimum in rank occur at $\gamma \approx 0.1$.

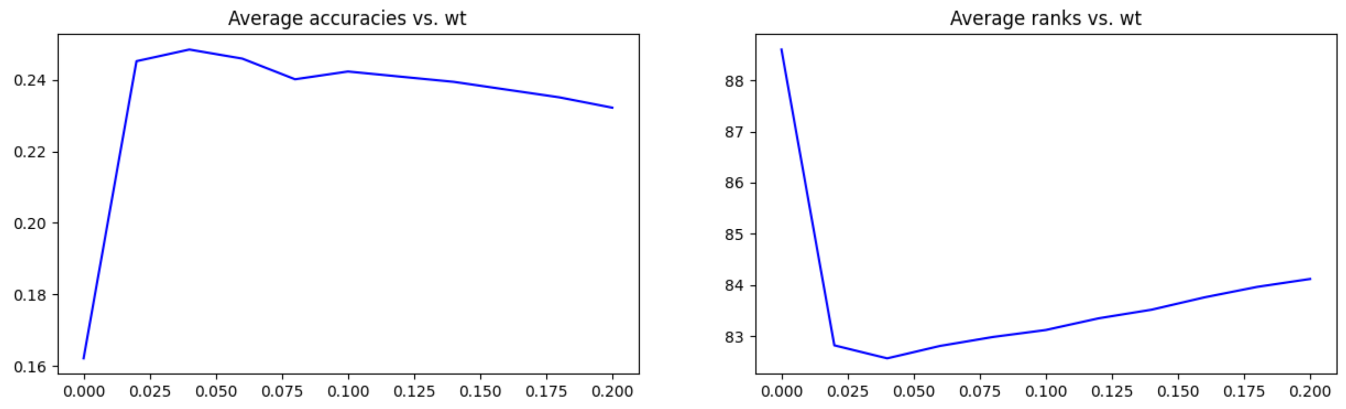


Figure 11: Accuracy and rank of the second approach of the NM method of Task 2 as γ is varied from 0 to 0.2 with a step of 0.02. A local maximum in accuracy and a local minimum in rank both occur at $\gamma = 0.04$, which gives the best performance.

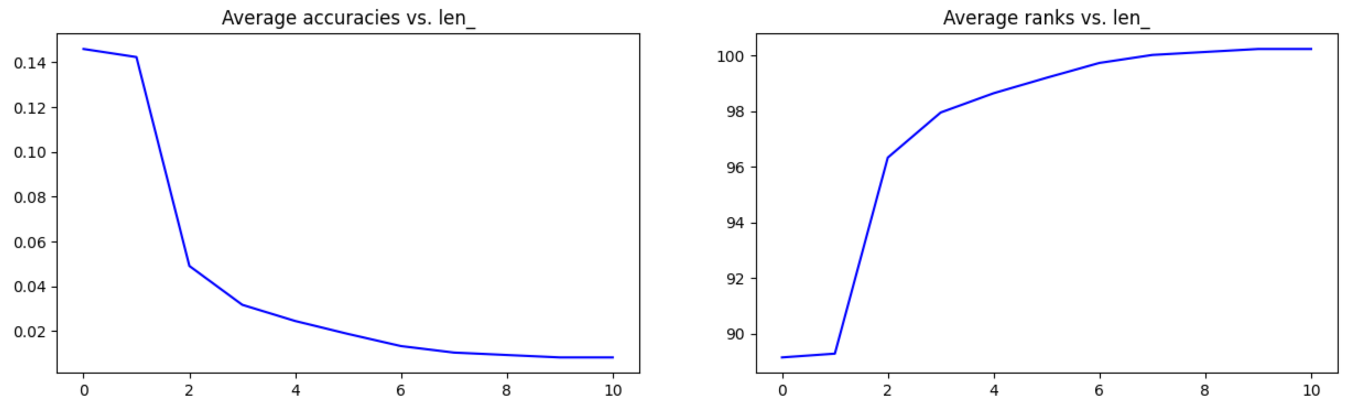


Figure 12: Accuracy and rank of the second approach of the eICF method of Task 2 as σ is varied from 0 to 10 with a step of 1. We observed that the results worsen at increased values of σ and that there is no local optimum position at both plots, so we concluded that this is not a good approach.

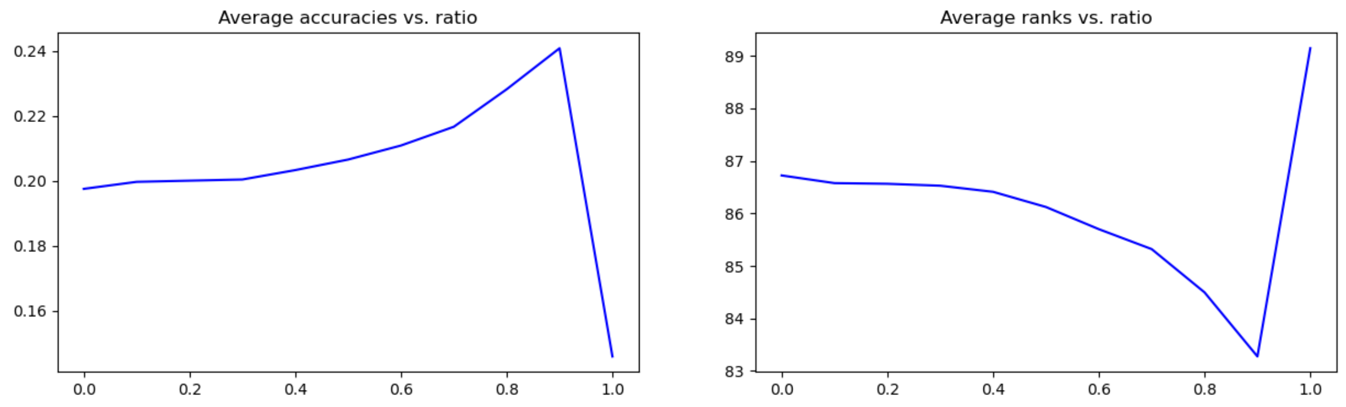


Figure 13: Accuracy and rank of the third approach of the eICF method of Task 2 as δ is varied from 0 to 1 with a step of 0.1. A local maximum in accuracy and a local minimum in rank both occur at $\delta \approx 0.9$.

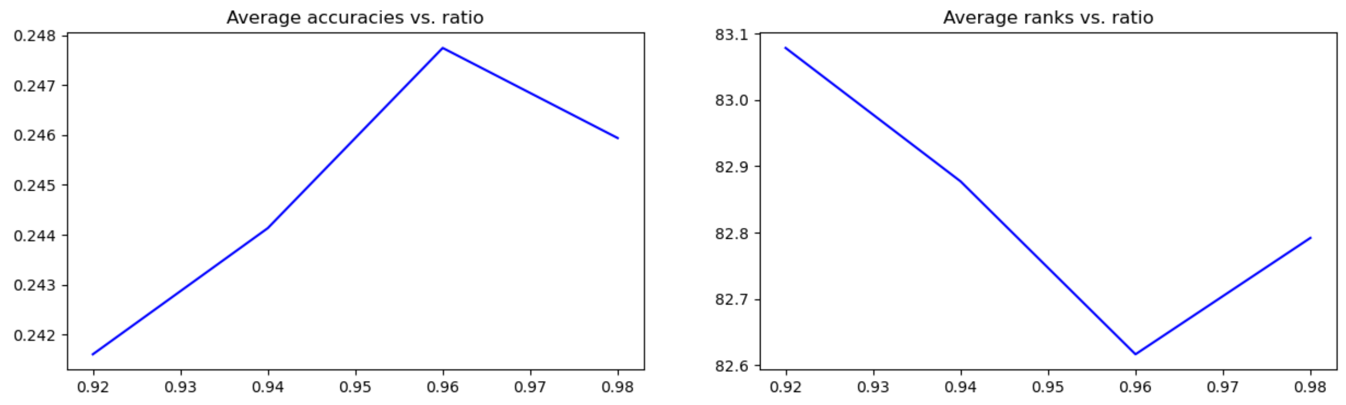


Figure 14: Accuracy and rank of the third approach of the eICF method of Task 2 as δ is varied from 0.92 to 0.98 with a step of 0.02. A local maximum in accuracy and a local minimum in rank both occur at $\delta = 0.96$, which gives the best performance.

for the clustering component of Segmented Frequent Itemsets, but there can be other clustering algorithms that are more aware of bipartite graph structures. Second, using the predictive results of one task can benefit the other; for example, one may perform alternating optimization between user-itemset graph completion and itemset-item graph completion while utilizing information from both graphs. Third, the notion of similarity in collaborative filtering approaches and adjusted neighboring occurrences in Neighbor Mining are defined in an unsophisticated manner in our approach, but considering more complex definitions of these notions may aid us in building models that better capture the relationship between itemsets (or between items), thus enhancing the model performance. Finally, the integration between user-item, user-itemset, and itemset-item datasets was not widely considered in our models, so taking this into account may reveal more hidden patterns behind the data. These issues might open up many future research venues.

REFERENCES

[1] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, "Collaborative filtering recommender systems," *The adaptive web: methods and strategies of web personalization*, pp. 291–324, 2007.

[2] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *2008 Eighth IEEE international conference on data mining.* Ieee, 2008, pp. 263–272.

[3] D. Bokde, S. Girase, and D. Mukhopadhyay, "Matrix factorization model in collaborative filtering algorithms: A survey," *Procedia Computer Science*, vol. 49, pp. 136–146, 2015.

[4] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 173–182.

[5] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural graph collaborative filtering," in *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, 2019, pp. 165–174.

[6] S. Rendle, W. Krichene, L. Zhang, and J. Anderson, "Neural collaborative filtering vs. matrix factorization revisited," in *Proceedings of the 14th ACM Conference on Recommender Systems*, 2020, pp. 240–248.

[7] Y. Dong, J. Li, and T. Schnabel, "When newer is not better: Does deep learning really benefit recommendation from implicit feedback?" *arXiv preprint arXiv:2305.01801*, 2023.

[8] C. Borgelt, "Frequent item set mining," *Wiley interdisciplinary reviews: data mining and knowledge discovery*, vol. 2, no. 6, pp. 437–456, 2012.

[9] 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*, vol. 2008, no. 10, p. P10008, 2008.

Approach	γ	Accuracy	Rank
1	–	**0.1676**	**88.610**
2	0.00	0.1622	88.601
2	0.02	0.2452	82.816
2	0.04	**0.2485**	**82.562**
2	0.06	0.2459	82.805
2	0.08	0.2402	82.980
2	0.10	0.2423	83.117
2	0.12	0.2409	83.345
2	0.14	0.2394	83.514
2	0.16	0.2373	83.755
2	0.18	0.2351	83.960
2	0.20	0.2322	84.114
2	0.30	0.2228	84.748
2	0.40	0.2178	85.178
2	0.50	0.2131	85.336
2	0.60	0.2109	85.585
2	0.70	0.2098	85.761
2	0.80	0.2084	85.860
2	0.90	0.2069	85.902
2	1.00	0.2055	85.961

Table 6: Task 2 evaluation on validation dataset for eISCF. The best result for each approach is highlighted.

[10] F. Parés, D. G. Gasulla, A. Vilalta, J. Moreno, E. Ayguadé, J. Labarta, U. Cortés, and T. Suzumura, "Fluid communities: A competitive, scalable and diverse community detection algorithm," in *Complex Networks & Their Applications VI: Proceedings of Complex Networks 2017 (The Sixth International Conference on Complex Networks and Their Applications).* Springer, 2018, pp. 229–240.

[11] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods.* SIAM, 1998.

Approach	σ	δ	Accuracy	Rank
1	–	–	**0.1459**	**89.142**
2	0	–	**0.1459**	**89.142**
2	1	–	0.1423	89.278
2	2	–	0.0491	96.322
2	3	–	0.0318	97.944
2	4	–	0.0246	98.636
2	5	–	0.0188	99.191
2	6	–	0.0134	99.724
2	7	–	0.0105	100.011
2	8	–	0.0094	100.119
2	9	–	0.0083	100.226
2	10	–	0.0083	100.226
3	–	0.00	0.1975	86.721
3	–	0.10	0.1997	86.575
3	–	0.20	0.2001	86.562
3	–	0.30	0.2004	86.525
3	–	0.40	0.2033	86.409
3	–	0.50	0.2066	86.120
3	–	0.60	0.2109	85.698
3	–	0.70	0.2167	85.319
3	–	0.80	0.2282	84.495
3	–	0.90	0.2409	83.276
3	–	0.92	0.2416	83.078
3	–	0.94	0.2441	82.877
3	–	0.96	**0.2477**	**82.616**
3	–	0.98	0.2459	82.792
3	–	1.00	0.1459	89.142

Table 7: Task 2 evaluation on validation dataset for eICF. The best result for each approach is highlighted.

A APPENDIX

A.1 Labor Division

The team performed the following tasks

- Implementation and evaluation of outfit recommendation algorithm [Nguyen]
- Generation the results of the test queries for outfit recommendation [Nguyen]
- Implementation of outfit generation algorithm [Wijaya]
- Generation the results of the test queries for outfit generation [Wijaya]
- Composition of progress report, final report, and final presentation [all]

A.2 Full disclosure wrt dissertations/projects

Wijaya: He is not doing any project or dissertation related to this project: his ongoing B.S. thesis is on AI fairness on drug repurposing applications.

Nguyen: He is not doing any project or dissertation related to this project: he has not composed his B.S. thesis.