[THESIS] BSc Econometrics and Operations Research

Rebuild Recency-aware Collaborative Filtering Recommender System and enhancement with Product Categories

Ngoc Diep Nhiep 524339

Erasmus School of Economics, Erasmus University Rotterdam

July 3, 2022

Supervisor: Luuk van Maasakkers

Second assessor: Jeffrey Durieux

Abstract

Recommender systems help optimize the purchasing process of E-Commerce. However, there are questions about the accuracy and efficiency of these techniques. This paper aims to recreate and confirm the results in the original paper about the Collaborative Filtering Recommender System by applying recency to enhance the standard recommendation system. The replicate outcomes agree with the original ones that popularity is a strong baseline, and the proposed recency-based model showed efficient performance as well as solutions to multiple existing problems such as cold-start or repetition and loyalty behaviors. However, as the difference in dataset size, the accuracy among distinguished best top-k recommendations might slightly differ. As an extension, this paper also aims to enhance the recommender system and extend the methodology with the contribution of Product Categories appearing as a new hyper-parameter in the formula. This new approach is promising in generalizing the shopping process and increasing the accuracy of the recommendations for a better customer journey.

Contents

1 Introduction

E-commerce has significantly risen in the past decades due to the substantial growth of internet usage and modern technology. The increase in internet access has led to the overload of data and information (Isinkaye et al., 2015). This situation not only hinders the customer journey (such as time-consuming to get access to online shopping) but also creates multiple challenges for the business to optimize their profit (Isinkaye et al., 2015; Costa and Macedo, 2013; Lu et al., 2015). Without prior knowledge and reliable reviews, for example, it is difficult and time-consuming for users to pick one among 17,000 movies on Netflix (Bennett et al., 2007), or even more challenging, 410,000 titles on Kindle only, not to mention the whole Amazon platform (Ekstrand et al., 2011). Information retrieval systems such as Google or DevilFinder could help solve parts of the problems; nevertheless, they did not prioritize and personalize the users' data to give the most appropriate suggestion (Isinkaye et al., 2015).

Researchers defined recommender systems as the agents of the personalized (recorded and processed) information to make recommendations based on users' preferences (Burke, 2007). They also assessed recommender systems as a program giving the most suitable suggestion for specific users by anticipating or "guessing" thanks to the information about their interest (Lu et al., 2015; Bobadilla et al., 2013). With this mechanism, it is believed to be the most valuable equipment to optimize the buying process in E-Commerce. Among all the innovative technologies, the Collaborative Filtering (CF) algorithm in recommender systems is the most successful and widely used (Deshpande and Karypis, 2004; Konstan et al., 1997). Ranging from such "IT big guys" (Netflix, Spotify, Amazon...) to small retail platforms, most commercial businesses nowadays have proceeded with the application of recommender systems. By and large, recommender systems bring advantages for service providers and users (Isinkaye et al., 2015; Pu et al., 2011).

CF algorithm in recommender systems appreciates each user's personalization and priority feature. This means that no matter how the user's profile and buying behavior are, the algorithm could offer the most suitable suggestions for him. For such a long time, buyers have mainly depended on and spent much time on unverified and uncertain "recommendations" such as word-of-mouth or recommendation letter (Resnick and Varian, 1997), meanwhile the scientific-based recommendation system is a personal and time-saving source for the users to consider and make the final decision when shopping online (Faggioli et al., 2020; Isinkaye et al., 2015). Moreover, the new item recommendations that the customer has never purchased before could bring diversity to each basket and help enhance their taste as well as the buying experience. With all these user benefits, a recommender system could be a potential solution for a user's

repetitive and loyal behavior, yet all users' unique and changeable behavior. More ideally, if a model works better than the other, a company could maximize its profit and do better than the other competitors in the same area (Palmatier and Sridhar, 2020).

Besides the advantages, there are also certain backwards of the recommender systems. These problems might come from the matter of information histories such as cold-start or a lack of fairness when recommending the popular items (Kumar and Sharma, 2016; Ricci et al., 2015). Without careful data processing, the insufficiency in dataset and item-user-order information would lead to the poor performance of the recommender systems. For details and reasons behind these factors that could degrade the recommender systems, we will explain and provide more in the section Literature Review. To overcome these obstacles, researchers have tried to apply various factors as implementation on the traditional recommender system, and some could be named as time, product taxonomy, sentiment analysis, or even social platform interactions of the users (Ziani et al., 2017; Hung, 2005; Ben-Shimon et al., 2007).

One in-depth research about top-k popular recommendation with a time limit (recency) that we would look into is the paper of Faggioli et al. (2020) "Recency Aware Collaborative Filtering for Next Basket Recommendation". This research aims to build a baseline model with popularity and apply recency in the collaborative filtering process. This is a leap from the standard method since it helps overcome the multiple challenges mentioned earlier of recommender systems. Here, our paper is an effort to recreate and solidify this study by inheriting different proposed methods to suggest items for the consumers, except for the FPMC* and T2V+Ada. As for the extension part, we target to extend the methodology to cover the product type of customers' baskets. Therefore, the main research question of this paper is **"How would a Recommender System change when including the product type of items in the process of calculating similarity?"**.

We have found out that for the replication part, the recency has been a great contributor to the performance of the enhanced algorithm. Even though our paper has slight differences with the accuracy value of top-k recommendation due to the different sizes of the dataset, the general conclusion stays the same as the paper of Faggioli et al. (2020). For the extension part, the product categories also positively contribute to the recommendation process. We introduce a new hyper parameter to represent the contribution of product type and aim at finding the optimal value; however, it is not yet certain which value of the hyper parameter would optimize the accuracy.

The paper is organized as follows. In the next section Literature Review, we will discuss studies of similar approaches in adding recency and product categories to the recommender

system. The following section is Data, which gives complete information about the dataset being used. Following is the Methodology - the rebuild from the original paper of Faggioli et al. (2020) and the extension with a contribution of product type. The next section is Application, explaining the evaluation method and detailed results of different methodologies performed with Python on Colab Research Google Notebooks. The last main section is the Conclusion, where we will summarize, self-evaluate the paper, and suggest future approaches and extensions.

2 Literature Review

2.1 Collaborative Filtering

As Herlocker et al. (2000) explained, the collaborative filtering (CF) systems suggest an individual by matching the information and recorded interests of that one to a group of people having the same behavior ("like-minded"). This helps confirm that the operation of recommender systems focuses more on the personal experience side of a customer journey (Mulvenna et al., 2000; Hung, 2005). To handle the enormous data of users, items, and the orders between them, CF does not analyze further into details and features of products but generates matrices to represent the relevance between users and items and to predict which items should be recommended (Wei et al., 2017). There are two main types of collaborative filtering: user-based and item-based. In user-based CF, recommendations are obtained by considering their ratings on items to calculate pairwise similarities of two arbitrary users. Meanwhile, item-based CF is the duality method of user-based, as it targets calculating the similarities of two arbitrary items instead of obtaining the suggestions (Tso-Sutter et al., 2008). These two classical and standard approaches for recommender systems CF are also the base of the original paper of Faggioli et al.. More details and mathematical notation of the methodology used could be found in the Methodology section.

2.2 Challenges in recommender systems

Theoretically, collaborative filtering (CF) in recommender systems has shown outstanding performance and high accuracy in suggesting items for users. However, in practical application, various factors would limit the quality of the recommendation framework. As mentioned before, we will take a closer look into other studies about the problems of recommender systems.

2.2.1 Cold-start

Collaborative filtering (CF) is the most popular approach to recommender systems that could utilize a large amount of history of purchases. However, it is also known for suffering from cold-start problems (Wei et al., 2017). Cold-start came when the distribution of recorded information and ratings from other users over all the items was not equal (Wei et al., 2017). In other words, "for all pairwise distance or similarity metrics, the item-based CF is unable to explore similarities between the items that have never been co-purchased but share the same neighborhoods nevertheless" (Deshpande and Karypis, 2004; Huynh, 2019). When a new user or a new item enters the data system, *cold-start* will occur as there is no information recorded in the data to support the recommendation process. For example, suppose that a new-added item is the most suitable choice for a user; the recommender has no confidence in suggesting that one. Consequently, the standard recommender system would not work correctly and accurately in this case.

2.2.2 Popularity bias

Besides the accuracy, there are also considerations for the fairness of recommender systems, specifically, the unfairness of popularity bias (Abdollahpouri, 2019). The popular items get recommender frequently to some specific users; meanwhile, multiple other choices are more related and relevant, yet being neglected by the recommender (Abdollahpouri, 2019; Huynh, 2019). Usually, a buyer has already decided and determined which item would best fit him. Hence they are unlikely to buy a popular yet irrelevant item. Moreover, the ignorance of long-tail (less popular) items might lose the businesses a chance to (1) get a fuller understanding of their users and, more important, (2) optimize their profit via the increase in purchases.

2.3 Time and Recency-based Collaborative Filtering Recommender System

Time has been a common factor to consider when implementing recommender systems. This is assessed as a "key aspect" of grocery recommendations since the purchases of a user could reveal the feature of seasonality (concept drifts) or even the changes in the taste of users (Faggioli et al., 2020). Usually, the standard recommender systems would treat any rating of a user for an item equally when calculating similarities without considering the real-time of those reviews. Meanwhile, it is the other case in reality: as one could change their taste over time, the most-recent rating would give the most valuable contribution when making a recommendation. Being aware of this, Ding et al. (2006) proposed the new framework that focused on the impact of time (in specific, recency) by applying weights for items based on their future preferences to calculate

expected accuracy. Another model from Nitu et al. (2021) also concerned a user's most recent traveling interest (via tweets and social platforms) and applied time-sensitive recency weight into the model. It outperforms most of the existing personalized recommendation systems in the area of travel suggestions. At this time of the decade, we have seen a boom in social networks and social media activities, most recency-based studies currently focus on the time, and users' activities on multiple social platforms as Larrain et al. (2015); Logesh and Subramaniyaswamy (2017); Huang et al. (2014).

Though including the time-sensitivity (recency) in the model shows significant efficiency, there has been little research on this area to update the recommendation system (Balloccu et al., 2022). The paper of Balloccu et al. (2022) specifically focuses on a single and detailed explanation for recommendation (such as the interaction of a user with an actress/actor profile on a movie website) instead of generally considering a metric with the combination of all explanatory factors (such as interaction with a movie having some actors/actresses). This, to some extent, agrees with the idea of the original paper by Faggioli et al. (2020) as considering only the most r recent basket (similar to the idea of considering the most recent interacted actor/actress by Balloccu et al. (2022)). Both of these experiments with time sensitivity included have shown the enhancement and increase in the quality of recommendations. Recency not only plays a role in the phase of consideration and purchase in the customer journey, but it could also appear in the phase of advocacy when customers give reviews and feedback. The CF algorithm with the inclusion of negative feedback (detected by negative meaning words and machine learning) was proved to have higher accuracy than the baseline method and out-perform both classic and state-of-the-art algorithms (Vinagre et al., 2015). With a similar mindset to concern about the online reviews and feedback, the inclusion of time (when) and time-decay (how) of social tagging in the CF algorithm has also shown a "substantial effect on accuracy" (Larrain et al., 2015). In general, most of the research on CF based on recency and time size limit has shown a significant improvement from the existing models.

2.4 Products type, Product Taxonomy, and Recommendation System

A few decades ago, the technological backward led to limited choices for the customers when shopping. However, the current situation is entirely upside-down; with a wide range of brands, sizes, colors, and prices, the customer might have difficulties making a final decision. As Cho and Kim (2004) have stated, the recommender system is a promising technology to overcome this headache for the customer. To enhance the model, researchers have taken a closer look at different perspectives of the products, such as product categories. Mild and Reutterer (2003)

came up with the extension of current CF to consider a multi-dimensional vector with a binary value representing users' choices when shopping (1 means bought and 0 means no). With this setting of notations, not only one but multiple purchased product categories could be assessed and considered. Other studies about recommender systems take product categories and types into account, such as papers of Albadvi and Shahbazi (2009); Lee and Hosanagar (2019). Besides categories, there is also research on the brand reputation of the products such as Hung (2005) took "brand name" characteristics into the model to highlight the "brand-sensitivity" in customer behavior.

The other approach is based on Product Taxonomy. This structure is viewed in the form of a tree to present the hierarchy. Hung (2005) explained the root node of the tree as all the products sold on the platform, with all the nodes connected to the root as the type of the products (such as clothes, cosmetics,...). Each product type has a bunch of lead nodes that contain further details: brands, sizes, prices, or even color. Even though it has a similar meaning to product type, the product taxonomy application is more complicated and has the feature of hierarchy to be applied to the data-mining method. From these simple explanations, we could observe that Product Type is a part of the Product Taxonomy, and together with the CF algorithm, we would only care about the type of product (first classification of the Product Taxonomy).

Concerning product type, it is often an area for investigating the marketing variables: from a tangible perspective such as promotion, price sensitivity, or quality of brands (Boyle et al., 2018) to hidden features such as choice behavior (Ainslie and Rossi, 1998). At first glance, it seems that the product categories are already unique and independent from each other, especially when thinking about the shopping behaviour; each consumer might already plan for their meal preparation and the needed groceries. However, it is not the case in the real world. There are correlations among product categories when analyzing user ratings; multiple studies have researched this assumption (Boyle et al., 2018).

3 Data

For replication part, we follow the original paper to use the "`Instacart`" dataset (Instacart.com, 2017). This originated from the Instacart website, which provides daily grocery delivery in the USA. This contains over 3 million observations as grocery orders from over 200 thousand users on the website. There is no specific date and time for each order, yet the baskets purchased sequence is provided. We would also keep using this data set for the Extension part to make a fair comparison concerning the performance of the original and extended methodology.

In this section, we would like to provide summary statistics for the data, together with steps

for the data cleaning. The Table 1 is the summary from original paper (Faggioli et al., 2020).

Table 1: Summary statistics of the full dataset

Dataset	#Users	#Items	Items selected	Baskets	min basket	min item	avg. baskets/user
`Instacart`	206209	49685	49685	3346083	2	10	16.22

We consider including the product categories as the `Department` in the dataset for the extension part. Below is the bar graph as the summary of the department with the number of items in a subset of the full dataset; the horizontal axis x is the `Department ID`, meanwhile for the vertical axis is y is the number of items for the corresponding index of the Department. The size of the subset considered here is 5% of the original dataset. The full explanation for the `Department ID` is in Section Appendix A.

This is the figure for the total number of items for each product category for comparison. As we can see, the product category as Department ID 16, which contains Diary products and eggs, shows the domination in the dataset with more than 500 products. Meanwhile, the department with the fewest items is Department 10 - Bulk. We observe that this data sparsity does not show balance and equal distribution. There is indeed a specific hierarchy in the number of items.

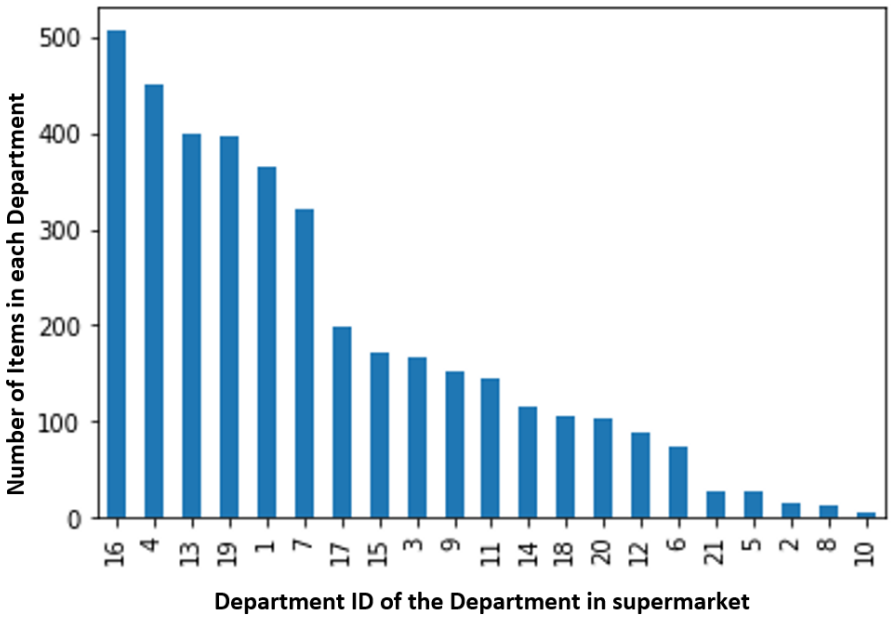


Figure 1: The departments in supermarket with corresponding number of items

4 Methodology

All the methods with terms, definitions, and formulas discussed below are inherited and extended from the original paper of Faggioli et al. (2020).

Let us denote that $\mathcal{U}$ is the set of n users and $\mathcal{I}$ is the set of m items. For each user u, the term for transactions as b_u^t where t indicates the ordinal position of the basket; hence b_u^1 would be the first basket (or transaction, interchangeably used) while $b_u^{B_u}$ would be the last one. We could also define $\mathcal{B}_u = \{b_u^t | t \in 1, ..., B_u\}$ as the set covering those baskets for a specific user u. Moreover, to verify the set of baskets of such user u containing a specific item i, we defined $\mathcal{B}_u^i = \{b_u^t | b_u^t \in \mathcal{B}_u \wedge i \in b_u^t\} \subseteq \mathcal{B}_u$, $|\mathcal{B}_u^i| = B_u^i$. The user-wise popularity for a given item i is formulated as:

$$\pi_i^u = \frac{B_u^i}{B_u}. \tag{1}$$

Hence, the global popularity which does not depend on any individual user could be defined as:

$$\pi_i = \frac{\sum_{u \in \mathcal{U}} B_u^i}{\sum_{u \in \mathcal{U}} B_u}. \tag{2}$$

4.1 Recency

This section provides more details and insights into the role of recency in the process. From the online lecture in UMAP '20: 28th ACM Conference by Faggioli (2020), recency could be informally understood as the *"last time the consumer bought that item"* or, in other words, as the number of most recent baskets bought by users. Faggioli et al. also pointed out that typically when buying groceries, there would be seasonality (the specific time of the year that the majority of customers buy specific product categories, for example, Christmas time and decorations products or Thanksgiving with turkey) and availability (whether those items are available or not). Without the recency, those could create drifts and outliers in the recommendation process. Therefore, by using the time frame limit, we could theoretically avoid the difficulties and give a better performance. The recency-involved user-wise popularity is defined as:

$$\pi_i^u @ r = \frac{\sum_{t=[B_u - r]_+}^{B_u} [\![i \in b_u^t]\!]}{min(r, B_u)}, \tag{3}$$

with r is the recency window size, $[\![x]\!]$ is the indicator function taking value 1 when x is true and 0 otherwise, and $[x]_+$ indicate the maximum between x and 0.

4.2 Popularity-based CF:

The standard popularity-based CF combines the weighted ratings from all users, and this mechanism works the same for both user-based and item-based CF. We have already introduced the way user-based and item-based CF work in Literature Review. While standard user-based CF would suggest items for user u depending on the ratings of a similar user, the proposed model of Faggioli et al. (2020) makes suggestions based on the popularity (either with recency or not) to replace the ratings. According to Faggioli et al. (2020), popularity, especially user-wise, is the most important and reliable indicator for grocery shopping. This could represent an item's level of needs and importance to a user. We inherited the build of the original paper for both user-based and item-based popularity-based CF.

4.3 Item popularity-based CF: IP-CF

This part will investigate the formation of methodologies for item-based CF with popularity. The formula for asymmetric cosine similarity is given in the paper of Faggioli et al. (2020). However, there is a small mistake in the original paper, as the coefficient for the probability needs to be switched. We performed and confirmed this based on the formula of conditional probability:

$$s(i,j) = \frac{|\mathcal{B}_i \cap \mathcal{B}_j|}{|\mathcal{B}_i|^{\alpha}|\mathcal{B}_j|^{1-\alpha}} = p(i|j)^{1-\alpha}p(j|i)^{\alpha}, \tag{4}$$

with $\mathcal{B}_i$ denoted the basket containing item i; the asymmetry coefficient α, working as the trade-off parameter to help the importance of the two probabilities, takes value in the range $[0, 1]$ (Faggioli et al., 2020). From this, we calculate the score for prediction as:

$$\hat{r}_i^u = \sum_{j \in \mathcal{J}} s(i,j)^q \pi_j^u, \tag{5}$$

with *user-wise popularity* π_i^u mentioned before implemented. In this formula, the *locality* coefficient q refers to the amount of nearest users/items really matter in the computation of similarity (Aiolli, 2013; Faggioli et al., 2020). Since the value of similarity $s(i,j)$ is in $[0,1]$, with higher value of locality, the similarity would be lower (Faggioli et al., 2020). In asymptotic case, when q approaches infinity ($q \to \infty$), $s(i,j) > 0$ if and only if item i is item j or two items ($i \neq j$) are always bought together (Faggioli et al., 2020).

In the original paper, the authors Faggioli et al. set the expectation that this method outperforms the standard user-wise popularity. There are two main reasons stated that: first, it could help adding diversity in recommendation, and secondly, there is the chance to increase

profits when the users buy suggested items from the recommender systems.

4.4 User popularity-based CF: UP-CF

As mentioned in Literature Review, user-based and item-based are the two duality cases in CF algorithm. In this section, we would investigate the formation of user-popularity-based. Firstly, the calculation of similarity between two users u and v which is also based on the conditional probability formula, is denoted as:

$$w(u,v) = \frac{|\mathcal{I}_u \cap \mathcal{I}_v|}{|\mathcal{I}_u|^\alpha |\mathcal{I}_v|^{1-\alpha}}, \tag{6}$$

with $\mathcal{I}$ denoted the set of items being bought by specific user (u and v in the formula), asymmetry coefficient α, and a *locality* coefficient q with the same definition given in the IP-CF similarity formula explanation.

Based on this similarity calculation and the same definition for the *locality*, we hereby give the definition of the prediction by Faggioli et al. (2020):

$$\hat{r}_i^u = \sum_{v \in \mathcal{U}} w(u,v)^q \pi_i^v, \tag{7}$$

with the same definition for *locality* coefficient q and the *user-wise popularity* π_i^u as mentioned before.

4.5 Recency-aware CF: UP-CF@r and IP-CF@r

Faggioli et al. (2020) new proposed model also implemented recency into the CF process via the popularity formula mentioned in Recency. The calculation for similarities based on the conditional probability formula of two users and two items stays the same as in the no-recency case. However, instead of the normal *user-wise popularity* π_i^u, we would apply the recency-based user-wise popularity $\pi_u^i@r$ from Recency. Therefore, we would expect an improvement in the prediction score due to the recency-based popularity.

The formula to calculate score of prediction for recency-involved item-popularity-based CF (IP-CF) is:

$$\hat{r}_i^u@r = \sum_{j \in \mathcal{J}} s(i,j)^q \pi_j^u@r, \tag{8}$$

and score of prediction of recency-involved user-popularity-based CF (UP-CF) is denoted as:

$$\hat{r}_i^u@r = \sum_{v \in \mathcal{U}} w(u,v)^q \pi_i^v@r. \tag{9}$$

4.6 Evaluation method

In this paper, we followed the evaluation method of the original paper for both the replication result and the extension by using normalized Discounted Cumulative Gain (nDCG). The length of the test basket is denoted by T_u. The evaluation measurement is based on the ranking, which would appreciate the results with a higher level of relevance. For example, to apply the measurement on top-k recommendations denoted as:

$$nDCG@k = \frac{1}{IDCG@k} \sum_{i=1}^{k} \frac{rel(R_{iu})}{log(i+1)}, \tag{10}$$

with

$$IDCG@k = \sum_{i=1}^{min(k,T_u)} \frac{1}{log(i+1)}. \tag{11}$$

The $rel(R_{iu})$ denoted graded relevance of the result at position i. The higher the relevance score, the more relevant the result to the user. The sum of graded relevance is called Cumulative Gain (CG), denoted as $\sum_p rel_{iu}$ with p as the position/rank of the result. The order of results would not change the final values of this CG value since it does not consider the rank (position) of the result yet. Next step, when calculating the full evaluation value of nDCG, each contributing value's order (rank) would be of great importance to the final result. There is a change from the original paper in the iDCG formula. Instead of taking the sum to k, the summation now is from the minimum value between k and the test set length. This could be seen as one of the small errors from the original paper, even though the author is already concerned about this problem in the recency-aware user-wise popularity formula. This might not cause a significant impact on the final result.

For our research and replication, we make use of the standard `nDCG package` with formula from `sklearn.metrics` (Järvelin and Kekäläinen, 2002; Wang et al., 2013; McSherry and Najork, 2008) for Python.

4.7 Product type enhancement Methodology

This paper focuses on the similarity affected by the product types of the items. The items might differ from each other. However, if they are of the same type, they share a certain similarity. From this perspective, we would like to implement the formula of the original cosine similarity in the paper of Faggioli et al. (2020). Denote the set $\mathcal{T}$ as the set of all the product types t

(or Departments as in the dataset **Instacart**). We also let t_i denote the product type of item i. We will also introduce a new parameter β to represent the similarity multiplier in case two items are from the same type.

The proposed formula for similarity including the product type becomes

$$s^*(i,j) = \frac{|\mathcal{B}_i \cap \mathcal{B}_j|}{|\mathcal{B}_i|^\alpha |\mathcal{B}_j|^{1-\alpha}} \times \beta^{\mathbb{1}\{t_i = t_j\}} = p(i|j)^{1-\alpha} p(j|i)^\alpha \times \beta^{\mathbb{1}\{t_i = t_j\}}, \tag{12}$$

with $s(i,j)$ as the enhanced similarity, β is the new parameter represents the level of contribution of product type, the indicator function ($\mathbb{1}\{t_i = t_j\}$) in the exponential of β is to check whether two arbitrary items are in the same department.

***Including Aisles of the products**: Knowing that if two products are in the same aisle, they must be in the same department, but not the other way around. We would denote hyper-parameter γ represent the contribution of aisle to the similarity between two item i and j:

$$s^*(i,j) = \frac{|\mathcal{B}_i \cap \mathcal{B}_j|}{|\mathcal{B}_i|^\alpha |\mathcal{B}_j|^{1-\alpha}} \times (\beta + \gamma^{\mathbb{1}\{a_i = a_j\}})^{\mathbb{1}\{t_i = t_j\}} = p(i|j)^{1-\alpha} p(j|i)^\alpha \times (\beta + \gamma^{\mathbb{1}\{a_i = a_j\}})^{\mathbb{1}\{t_i = t_j\}}, \tag{13}$$

$$\mathbb{1}\{t_i = j_j\} = \begin{cases} 0, & \text{if } t_i \neq t_j \\ 1, & \text{if } t_i = t_j. \end{cases}$$

The similar setup for indicator function of aisles, value 1 when two items are in the same aisle, 0 otherwise. If the indicator function for aisle takes 1 as the value, the indicator function for the department must take 1 as well. However, for the similarity value, we would like to normalize them into the range of [0,1] where 0 represents the extreme case that there is no similarity and 1 represents the case that the two items are identical. Adding β requires the normalization step for the new similarity. The normalization formula for similarity with department implementation is:

$$s^*(i,j)_{normalized} = \frac{s^*_(i,j) - s^*(i,j)_{min}}{s^*(i,j)_{max} - s^*(i,j)_{min}},$$

and knowing that the maximum value for new similarity (s^*) between two items is β while the minimum value is 0, we could simplify the formula to:

$$s^*(i,j)_{normalized\_department\_aisle} = \frac{s^*(i,j)}{(\beta + \gamma)}. \tag{14}$$

5 Application

5.1 Data preparation

The total observations of the **Instacart** dataset is 3,000,000 (million) orders, which is enormous for a personal laptop. For a better performance without losing any generalization, the paper will only use 5% of the dataset. Half of the last basket (n^{th} basket) is for the validation, and the other half is for the testing process; the remaining baskets are for training. Below is the figure that illustrates the data split for the training-testing-validating process.

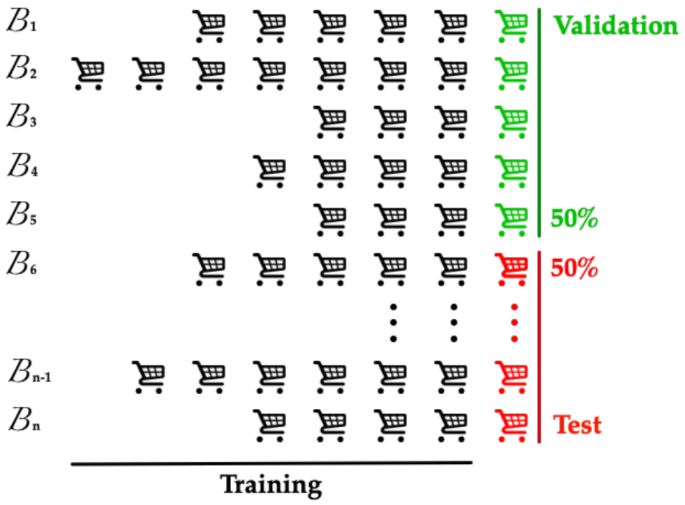


Figure 2: This figure illustrates the training, evaluation, and test set. Black baskets denote the training set; the green baskets are for validation, and the red ones are the test set. This follows the guidance to use the orders from 1 up to (n-1). For the last order, we split half-half. (Faggioli et al., 2020)

5.2 Recency-aware result

We run the code with different k for top-k recommended products to the user, with k= 5, 10, and B (the length of the test set). To obtain these findings below, we use Python with environment of Google Colab Notebook 12GB RAM on 64-bit PC with Intel(R) Core(TM) i5-7200U.

There is a set of hyper-parameters with different values (taken from the original paper for a fair comparison) for testing:

- **Recency window**: $r \in \{1, 5, 25, 100, \infty\}$;
- **Locality**: $q \in \{1, 5, 10, 50, 100, 1000, \infty\}$;
- **Asymmetry**: $\alpha \in \{0, 0.25, 0.5, 0.75, 1\}$.

For the model evaluation, we perform on the test set and compare the three enhanced recency-aware with four baseline models: GPop (global population k best items), IP-CF, UP-CF, and UW-Pop. Each of these models is explained with details in Methodology. The obtained

outcomes for comparison would be the level of accuracy and relevance when recommending 5, 10, or B products for the user. Those are calculated based on the nDCG@k also explained in the Methodology. The following table only reports the highest outcome over all the above combinations of hyper-parameter values with three values of k being considered.

Table 2: Result from Faggioli et al. (2020) on `Instacart`'s dataset.

Algorithm	nDCG@5	nDCG@10	nDCG@B
gPop	0.098	0.109	0.081
UWPop	0.406	0.387	0.327
IP-CF	0.256	0.256	0.206
UP-CF	0.166	0.147	0.123
UWPopr	0.415	0.395	0.336
IP-CFr	0.427	0.408	0.346
UP-CFr	**0.429**	**0.411**	**0.349**

Table 3: Replication result on 5% subset of `Instacart`'s dataset.

Algorithm	nDCG@5	nDCG@10	nDCG@B
gPop	0.213	0.215	0.165
UWPop	0.629	0.639	0.605
IP-CF	0.629	0.6406	0.574
UP-CF	0.633	0.646	0.575
UWPopr	0.634	**0.645**	0.614
IP-CFr	0.636	0.648	0.576
UP-CFr	**0.6409**	**0.659**	**0.578**

5.2.1 Comparison with baseline models

This section will compare the proposed models against baseline ones, based on the results when considering different values of k, in turn, 5, 10, and length of test size B. These values for evaluation represent the "available space" of different devices. The top 5 items would represent a phone screen, the top 10 as a laptop screen, while the top B items are the extreme case to print all the total number items equal to the length test set if known (Faggioli et al., 2020). The higher the results in the table, the more accurate and relevant the suggestions for the users. Faggioli et al. have mentioned these results as the satisfaction level of the customers. However, this might not be the correct way to interpret it. For example, when a recommender system gave three (3) items as suggestions for the users, and they bought two (2) out of them, the buying rate is higher than 60%; meanwhile, with ten (10) items, if they bought 4 out of them, the rate is only 40%. Nevertheless, we could not assume that giving three (3) items is more satisfying for customers since they bought more items in quantity in the second case. This is why from our perspective, the nDCG@k with corresponding values only represents the algorithm's relevance and accuracy, not the users' personal experience.

These findings agree with the original paper that the extreme case with B items brought the least matching and relevant experience for the user. The methodology could explain the reason behind this, as with B items, iDCG is maximum and makes the guessing process harder since most of the items are penalized. Regarding the best performance of how many k items to print, Faggioli et al. stated that nDCG@5 with printing on phone screen performed the best out of three options. However, we found that printing ten items gain the highest accuracy and

relevance, except for the global popularity baseline method.

From the result of the original paper, UP-CF@r saw the stability of giving the highest results highlighted in the table 2. Following this method, the other two proposed models of Faggioli et al. also give potential outcomes. These promising findings have been explained by the "high performances" of user-wise popularity (Faggioli et al., 2020). From our research result, we also observed the same situation. Our replicated UP-CF@r also obtained the highest value of nDCG@k, followed by the other two proposed recreated models: IP-CF@r and UWPop@r. Another remarkable finding is that recency contributes to and improves the recommendation process. The proposed models outperform and increase the accuracy compared to the baseline and standard models.

5.2.2 Hyper-parameter evaluation

In the original paper, Faggioli et al. performed the test on two different datasets, which led to the different values of hyper-parameters for the optimal final result. In this paper, we only test on the `Instacart`'s dataset. The two following tables give an overview of results when changing the recency values to run the UWPop@r method.

Figure 3: nDCG@B of UWPop@r with r=5

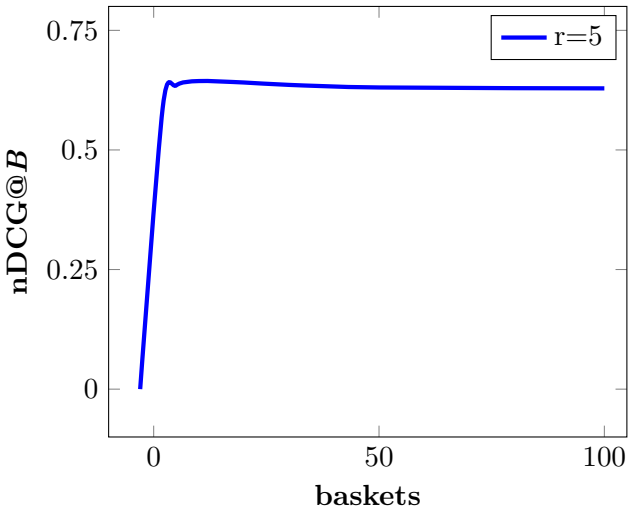
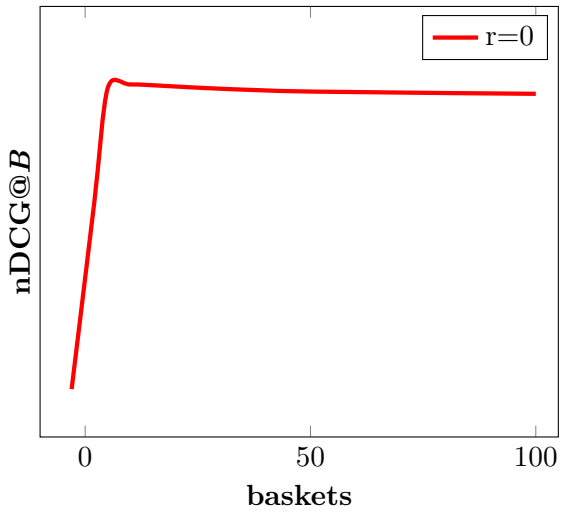


Figure 4: nDCG@B of UWPop@r with r=0



In terms of hyper-parameter values validation, for the non-recency ($r = 0$) case, the overall highest values are obtained when $\alpha = 1$ and $q = 1$ while if considering recency, the best to get is from (recency) $r = 5, \alpha = 1$, and $q = 5$. Both cases with and without recency show that the locality is preferred to have low value. Without a limit on the number of time-sensitive baskets when shopping (r=0), q=1 worked best. Meanwhile, when recency is set to optimal value 5,

the locality 5 works best. This also agrees with the original paper's results since Faggioli et al. also concluded the system preference of low q since most of the algorithms preferred $q = 5$ or $q = 10$ in the original paper. When testing on the dataset `Instacart`, Faggioli et al. assessed that it showed fairly stable results. This is true for asymmetry value α since in all cases $\alpha = 1$ gives the best outcomes. For recency, of all the values mentioned above in the parameter setup, the best outcomes are obtained with recency equals 5, which means when recommending items, recommender systems work best to consider the latest five baskets of a user.

We would like to add a side-note to explain the minor difference in results between the original paper and ours. For the evaluation method nDCG and even the consine similarity calculation, the authors build up the formula quite differently from the existing and standard formula of packages used when we replicate the codes. Moreover, with the limitation in technology and skills, we entirely use the packages. Details, references, and authors of these, such as package *similaripy* or *nDCG score from Sci-kit learning* could be found in the Appendix with the coding summary.

5.3 Product type and aisle involved result

We have included a hyper parameter for product type in the similarity of IP-CF (with and without recency) from the extension methodology. The extension experiments' steps required creating a huge matrix as the indicator matrix for parameter β representing the product categories. Due to computer storage and RAM capacity, we would consider only a total subset of 0.005, which is ten times less than the replication part dataset. However, this is expected not to affect the generality of the experiment. We are aware that we could obtain more reliable results with more products and users. We have tried to approach with top 5000 popular products as well. However, since there is a mismatch in matrix multiplication, we will attach it to the Appendix but not analyze the full results of that effort. For a fair comparison, we would also re-run the testing models (such as UWPop@r, IP-CF(@r), and UP-CF(@r)) on a new subset of data.

The product type in this the data set `Instacart` is the `Department` with specific `DepartmentID` in the file about all products. We could also extend to another level with aisles; details are already discussed in the Methodology section. Due to the limitations of time and capacity, we will only analyze the implementation with departments of the products. For the aisle implementation, which could be potential future research, we could denote the hyper-parameter γ to represent aisles with suitable values to test.

For the experiments on Departments (Product types) now, we will perform on testing set.

The setup for hyper parameters is as follows: we keep testing with the mentioned values of recency, locality, and asymmetry. For the new product type level - β, we propose a set of value: $\beta \in \{1, 2, 5, 8, 10, 15\}$.

Table 4: Extension result with the product types (department) including

Algorithm	nDCG@5	nDCG@10	nDCG@B
GPop	0.2405	0.2378	0.177
UWPop	0.495	0.496	0.477
IP-CF	0.389	0.498	0.479
IP-CF new	0.530	0.583	0.593
UWPop@r	0.466	0.471	0.441
IP-CF@r	0.554	0.595	0.613
IP-CF@r new	0.403	0.561	0.583

Table 4 reports both the original models' results and enhanced ones with a subset of 0.005 for a fair comparison. The values are the highest when using different values of β for testing. We chose such set of values for β to compare the level of impact between the department (product type) and the other hyper-parameters such as recency or locality. Within the same scope, the values for testing are reasonable. After all the combinations of different values, the outcomes with $\beta = 5$ show the highest score for recommendations to the user.

The enhanced algorithms generally show slightly better performance than the original ones, so they also outperform the baseline model. Since we do not test on UP-CF because of the irrelevance with the department, the IP-CF with and without recency has the best performance. The enhanced algorithm IP-CF (without recency) make an improvement from old IP-CF with the gap of, respectively, 0.141 for k=5, 0.085 for k=10, and 0.114 for $k = B$ (length of the test set). Nevertheless, it is not the case when we also consider recency. When including both product types and recency, the accuracy decreases from the original ones. Another remarkable observation is that nDCG@B shows the highest accuracy and relevance, which is completely different from the originally proposed models. We assessed this as a sign of the lack of reliability in these results as it shows an upgrade in some perspectives, yet the imbalance in values between testing and validating set because of sample size might cause serious mistakes in the results. Normally, the results between the test set and the validation set should show similarities. Meanwhile, in this study case, it shows a huge gap between the two values each iteration.

In terms of parameter value validation, the findings also agree with ones from the experiments in the replication part, as $\alpha = 1$ and q =5 with $recency = 5$ are still preferred the most. Therefore, we could prove that the recency with value 5 is the optimal value for the hyper-

parameter. However, the efficiency when combining recency and product type does not live up to our expectations. We aim to enhance the proposed models, but these results suggest that it is better to choose either limit in most recent basket quantities or the history and categories of customers' buying information.

6 Conclusion

In this paper, we have rebuilt the existing Collaborative Filtering Recommender System based on the original paper, with popularity as a baseline and recency in shopping behavior as the new approach. Moreover, we have come up with the extension about product type to implement the existing models with the main research question: **"How would a Recommender System change when including the product type of items in calculating similarity?"**. Our findings from the recreation part agree with most of the crucial outcomes of the original paper. Although there are some minor differences between the two papers' results, it does not affect the main general conclusion from the studies that recency-based with popularity proposed models outperform the baseline models, show a significant improvement from the standard recommender systems and achieve an "state-of-the-art" performance. Among all the proposed models, UP-CF with and without recency shows the best performance. Following with the two proposed models, IP-CF(@r) and UWPop(@r).

For the extension part, we propose two new models to enhance the recommender system with the department and aisle of the items, which belongs to the product type and taxonomy area. There is already research about the hierarchy of products (from general to specific items), yet our contribution and effort to include different department types is to update and improve the current recommender system. We have achieved promising results when including only product types with a noticeable increase in accuracy and score. However, the results are not as expected when trying to integrate both recency and the product type. This could be a suggestion to use either recency with popularity or product type with the popularity of an item to get the optimal results.

A few limitations of this study include that we did not manage to make a proper and optimal data processing for the extension part. The insufficiency of data subsets has led to the reliability problems of results, even though they are promising to apply in reality. Specifically, the matter of reliability here stays in the imbalance between testing and validation test results and the inconsistency of nDCG@k ordering. Another minor limitation is that using only one data set could not capture the generality of the research. We only observe the potential on one data set, not a wide range of different datasets. Lastly, the lack of replication for FMPC and Ada

approach is a missing link to observing how such proposed models improve from the current advanced models.

Further research on this research could address the aforementioned limitations. Regarding the data processing, we have already made an effort to take only top popular products with their corresponding users and baskets. With this approach, not only is the sample size assured sufficiency, but it could also prioritize the popularity of the recommendation process. As the popularity is one of the essential criteria in this study, the top-popular subset could be the promising and correct direction. An additional suggestion is to inherit the methodology built with aisle and extend to see the correlation of department and aisle among items. This could be an interesting topic that can be applied to multiple datasets. IF further research could adapt those methodologies and expand with more advanced ensemble methods, it could be a promising tool and solution for existing recommender systems.

7 Acknowledgments

I would like to thank Luuk van Maasakkers for his guidance and support during the writing process of this paper. I have learned a lot from his advice for research direction and extension of the studies. I am thankful for all the feedback, discussions, and support from the supervisor, proofreaders, family, and friends.

References

Abdollahpouri, H. (2019). Popularity bias in ranking and recommendation. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 529–530.

Ainslie, A. and Rossi, P. E. (1998). Similarities in choice behavior across product categories. *Marketing Science*, 17(2):91–106.

Aiolli, F. (2013). Efficient top-n recommendation for very large scale binary rated datasets. In *Proceedings of the 7th ACM Conference on Recommender Systems*, pages 273–280.

Albadvi, A. and Shahbazi, M. (2009). A hybrid recommendation technique based on product category attributes. *Expert Systems with Applications*, 36(9):11480–11488.

Balloccu, G., Boratto, L., Fenu, G., and Marras, M. (2022). Post processing recommender systems with knowledge graphs for recency, popularity, and diversity of explanations. *arXiv preprint arXiv:2204.11241*.

Ben-Shimon, D., Tsikinovsky, A., Rokach, L., Meisles, A., Shani, G., and Naamani, L. (2007). Recommender system from personal social networks. In *Advances in intelligent web mastering*, pages 47–55. Springer.

Bennett, J., Lanning, S., et al. (2007). The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York.

Bobadilla, J., Ortega, F., Hernando, A., and Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-based systems*, 46:109–132.

Boyle, P. J., Kim, H., and Lathrop, E. S. (2018). The relationship between price and quality in durable product categories with private label brands. *Journal of Product & Brand Management*.

Burke, R. (2007). Hybrid web recommender systems. *The adaptive web*, pages 377–408.

Cho, Y. H. and Kim, J. K. (2004). Application of web usage mining and product taxonomy to collaborative recommendations in e-commerce. *Expert systems with Applications*, 26(2):233–246.

Costa, H. and Macedo, L. (2013). Emotion-based recommender system for overcoming the problem of information overload. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pages 178–189. Springer.

Deshpande, M. and Karypis, G. (2004). Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, 22(1):143–177.

Ding, Y., Li, X., and Orlowska, M. E. (2006). Recency-based collaborative filtering. In *Proceedings of the 17th Australasian Database Conference-Volume 49*, pages 99–107.

Ekstrand, M. D., Riedl, J. T., Konstan, J. A., et al. (2011). Collaborative filtering recommender systems. *Foundations and Trends® in Human–Computer Interaction*, 4(2):81–173.

Faggioli, G. (2020). Recency aware collaborative filtering for next basket recommendation remote presentation 2020. Accessed from `https://www.youtube.com/watch?v=WI1P6nKGwlg` on June 29 2022.

Faggioli, G., Polato, M., and Aiolli, F. (2020). Recency aware collaborative filtering for next basket recommendation. In *Proceedings of the 28th ACM Conference on User Modeling, Adaptation and Personalization*, pages 80–87.

Herlocker, J. L., Konstan, J. A., and Riedl, J. (2000). Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 241–250.

Huang, C.-L., Yeh, P.-H., Lin, C.-W., and Wu, D.-C. (2014). Utilizing user tag-based interests in recommender systems for social resource sharing websites. *Knowledge-Based Systems*, 56:86–96.

Hung, L.-p. (2005). A personalized recommendation system based on product taxonomy for one-to-one marketing online. *Expert systems with applications*, 29(2):383–392.

Huynh, T. H. (2019). Basket-sensitive random walk and factorization machine recommendations for grocery shopping.

Instacart.com (2017). The instacart online grocery shopping dataset 2017. Accessed from `https://www.instacart.com/datasets/grocery-shopping-2017` on June 29 2022.

Isinkaye, F. O., Folajimi, Y. O., and Ojokoh, B. A. (2015). Recommendation systems: Principles, methods and evaluation. *Egyptian informatics journal*, 16(3):261–273.

Järvelin, K. and Kekäläinen, J. (2002). Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446.

Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., and Riedl, J. (1997). Grouplens: Applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87.

Kumar, B. and Sharma, N. (2016). Approaches, issues and challenges in recommender systems: a systematic review. *Indian journal of science and technology*, 9(47):1–12.

Larrain, S., Trattner, C., Parra, D., Graells-Garrido, E., and Nørvåg, K. (2015). Good times bad times: A study on recency effects in collaborative filtering for social tagging. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 269–272.

Lee, D. and Hosanagar, K. (2019). How do recommender systems affect sales diversity? a cross-category investigation via randomized field experiment. *Information Systems Research*, 30(1):239–259.

Logesh, R. and Subramaniyaswamy, V. (2017). Learning recency and inferring associations in location based social network for emotion induced point-of-interest recommendation. *Journal of Information Science & Engineering*, 33(6).

Lu, J., Wu, D., Mao, M., Wang, W., and Zhang, G. (2015). Recommender system application developments: a survey. *Decision Support Systems*, 74:12–32.

McSherry, F. and Najork, M. (2008). Computing information retrieval performance measures efficiently in the presence of tied scores. In *European conference on information retrieval*, pages 414–421. Springer.

Mild, A. and Reutterer, T. (2003). An improved collaborative filtering approach for predicting cross-category purchases based on binary market basket data. *Journal of Retailing and consumer Services*, 10(3):123–133.

Mulvenna, M. D., Anand, S. S., and Büchner, A. G. (2000). Personalization on the net using web mining: introduction. *Communications of the ACM*, 43(8):122–125.

Nitu, P., Coelho, J., and Madiraju, P. (2021). Improvising personalized travel recommendation system with recency effects. *Big Data Mining and Analytics*, 4(3):139–154.

Palmatier, R. W. and Sridhar, S. (2020). *Marketing strategy: Based on first principles and data analytics*. Bloomsbury Publishing.

Pu, P., Chen, L., and Hu, R. (2011). A user-centric evaluation framework for recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 157–164.

Resnick, P. and Varian, H. R. (1997). Recommender systems. *Communications of the ACM*, 40(3):56–58.

Ricci, F., Rokach, L., and Shapira, B. (2015). Recommender systems: introduction and challenges. In *Recommender systems handbook*, pages 1–34. Springer.

Tso-Sutter, K. H., Marinho, L. B., and Schmidt-Thieme, L. (2008). Tag-aware recommender systems by fusion of collaborative filtering algorithms. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1995–1999.

Vinagre, J., Jorge, A. M., and Gama, J. (2015). Collaborative filtering with recency-based negative feedback. In *Proceedings of the 30th annual ACM symposium on applied computing*, pages 963–965.

Wang, Y., Wang, L., Li, Y., He, D., Chen, W., and Liu, T.-Y. (2013). A theoretical analysis of ndcg ranking measures. In *Proceedings of the 26th annual conference on learning theory (COLT 2013)*, volume 8, page 6.

Wei, J., He, J., Chen, K., Zhou, Y., and Tang, Z. (2017). Collaborative filtering and deep learning based recommendation system for cold start items. *Expert Systems with Applications*, 69:29–39.

Ziani, A., Azizi, N., Schwab, D., Aldwairi, M., Chekkai, N., Zenakhra, D., and Cheriguene, S. (2017). Recommender system through sentiment analysis. In *2nd international conference on automatic control, telecommunications and signals*.

Appendix A

Table 5: Details of product type of the dataset

Department ID	Meaning of the department number
Department 1	Frozen products (frozen)
Department 2	Other products
Department 3	Bakery
Department 4	Produce
Department 5	Alcohol
Department 6	International
Department 7	Beverages
Department 8	Pets
Department 9	Dry goods pasta
Department 10	Bulk
Department 11	Personal care
Department 12	Meat, seafood
Department 13	Pantry
Department 14	Breakfast
Department 15	Canned goods
Department 16	Dairy eggs
Department 17	Household
Department 18	Babies
Department 19	Snacks
Department 20	Deli
Department 21	Missing / Other

Appendix B

Packages and Initialization

```
!pip install similaripy

# Imports
import os
import time
import numpy as np
```

```
import pandas as pd
# In order to deal with large sparse matrix we need to compresse them using
# the sparse sub_module of scipy lib
from scipy import sparse
import scipy.sparse as sps
from sklearn.metrics import ndcg_score
import similaripy as sim
```

Data processing in one function

```
def instacartProcess(dataPath, item_threshold=10, basket_threshold=2,
    subdata=subdata):
  # Read products.csv
  df_products = pd.read_csv(os.path.join(dataPath,"products.csv"))
  df_products.columns = ['PID', 'description', 'categoryId', 'departmentId'] #name
      the columns
  # Read departments.csv and merge
  tmp = pd.read_csv(os.path.join(dataPath,"departments.csv"))
  tmp.columns = ['departmentId', 'department']
  df_products = pd.merge(df_products, tmp, on='departmentId')
  # Read aisles.csv and merge
  tmp = pd.read_csv(os.path.join(dataPath,'aisles.csv'))
  tmp.columns = ['categoryId', 'category']
  df_products = pd.merge(df_products, tmp, on='categoryId')[['PID',
      'description','department','category']]
  del tmp

  # preprocessing

  df_order_products_prior =
      pd.read_csv(os.path.join(dataPath,"order_products__prior.csv"))
  df_order_products_train =
      pd.read_csv(os.path.join(dataPath,"order_products__train.csv"))
  df_order_products = pd.concat([df_order_products_prior,
      df_order_products_train])[['order_id', 'product_id']]
  df_order_products.columns= ['BID','PID']
  del df_order_products_prior, df_order_products_train,
```

```
# Removing all items that appears in less than item_threshold baskets
products_count = df_order_products['PID'].value_counts()
df_order_products=
    df_order_products.loc[df_order_products['PID'].isin(products_count[products_count
    >= item_threshold].index)]
del products_count
# Updating production list
pd.merge(df_products,df_order_products['PID'],on='PID')

df_orders = pd.read_csv(os.path.join(dataPath,"orders.csv"))[['order_id',
    'user_id', 'order_number', 'eval_set']]
df_orders.columns = ['BID','UID','order', 'set']

# User filtring
# Removing users with less than basket_threshold baskets
user_count = df_orders['UID'].value_counts()
user_filter = user_count[(user_count >= basket_threshold) &
    (np.random.rand(len(user_count))< subdata)] #choose a subset of the user
del user_count
df_orders = df_orders[df_orders['UID'].isin(user_filter.index)] #choose all the
    baskets of the user
del user_filter



user_dict = dict(zip(df_orders['UID'].unique(),
    range(len(df_orders['UID'].unique()))))
df_orders['UID'] = df_orders['UID'].map(user_dict)
del user_dict

# reset product index
df_order_products =
    df_order_products.loc[df_order_products['BID'].isin(df_orders['BID'].unique())]
df_products =
    df_products[df_products['PID'].isin(df_order_products['PID'].unique())]
product_dict =
    dict(zip(df_order_products['PID'].unique(),range(len(df_order_products['PID'].unique()))))
df_products['PID'] = df_products['PID'].map(product_dict)
df_order_products['PID'] = df_order_products['PID'].map(product_dict)
del product_dict
```

```
  # Join Tables

  df_data = pd.merge(df_orders, df_order_products, on= 'BID')
  del df_orders, df_order_products

  # Setting last baskets as dev/test sets
  last_basket_indexes =
      df_data.iloc[df_data.groupby(['UID'])['order'].idxmax()]['BID'].values
  df_data.loc[df_data['BID'].isin(last_basket_indexes),'set']='test'
  df_data.loc[df_data['set']=='prior', 'set'] = 'train'
  del last_basket_indexes

  # train test split data
  df_split =
      df_data[df_data['set']=='test'].groupby(by=['UID'])['PID'].apply(list).reset_index(name='artic
  msk = (np.random.rand(len(df_split))<0.5)
  df_dev, df_test = df_split[msk], df_split[~msk]
  del df_split

  df_train = df_data[df_data['set']=='train'][['UID','BID','order','PID']]
  dev_set = dict(zip(df_dev['UID'],df_dev['articles']))
  test_set = dict(zip(df_test['UID'],df_test['articles']))

  del msk, df_test, df_dev, df_data

  return df_train, dev_set, test_set, df_products
```

Top-n, prediction, evaluation function

```
def top_n(row, n): #this is for the top-n recommendations, we would plug in k later
    # Get user indices to sort the given row
    top_indices = row.argsort()[-n:][::-1] #return the top n of the indices
    # Use the top_indices to get top_values score
    top_values = row[top_indices] #return the corresponding idea
    return top_values, top_indices

def prediction(predMat, k): #this is to give the value for the prediction
```

```
    '''
    In :
      predMat : the prediction matrix
      {UWPop, UB-CF, IB-CF or gpop}with/out recency (@r)
    Out :
      score, pred : ndarray of shape =(n_users, k)
      retun the top-k score and prediction matrix
    '''
    n_users = predMat.shape[0]
    score = np.zeros((n_users, k))
    pred  = np.zeros((n_users, k))
    for i in range(n_users):
        score[i], pred[i] = top_n(predMat[i],k)
    return score.astype('float64'), pred.astype('int64')

def evaluation(score, pred, test_set, dev_set, k):
    # Get the test and dev set User IDs
    test_keys = test_set.keys()
    dev_keys = dev_set.keys()
    # Construct the True_relecvance and score vectors
    true_relevance_test = np.asarray([np.isin(pred[key],test_set[key]).astype(int)
        for key in test_keys])
    true_relevance_dev = np.asarray([np.isin(pred[key],dev_set[key]).astype(int) for
        key in dev_keys])
    score_test = score[list(test_keys)]
    score_dev = score[list(dev_keys)]
    # Calculate the ndgc@k evaluation metric
    test_ndcg_score = ndcg_score(true_relevance_test, score_test, k=k)
    dev_ndcg_score = ndcg_score(true_relevance_dev, score_dev, k=k)
    return test_ndcg_score, dev_ndcg_score

def evaluationGlobal(pred, test_set, dev_set, k):
    # Get the test and dev set User IDs
    test_keys = test_set.keys()
    dev_keys = dev_set.keys()
    # Construct the True_relecvance and score vectors
    true_relevance_test = np.asarray([np.isin(pred[key],test_set[key]).astype(int)
        for key in test_keys])
```

```
    true_relevance_dev = np.asarray([np.isin(pred[key],dev_set[key]).astype(int) for
        key in dev_keys])
    pred_test = pred[list(test_keys)]
    pred_dev = pred[list(dev_keys)]
    # Calculate the ndgc@k evaluation metric
    test_ndcg_score = ndcg_score(true_relevance_test, pred_test, k=k)
    dev_ndcg_score = ndcg_score(true_relevance_dev, pred_dev, k=k)
    return test_ndcg_score, dev_ndcg_score
```

Baseline model:

Evaluation of GPop:

```
n_items = df_products['PID'].unique().shape[0]
predMat = GPopMat(df_train, n_items, k)
score, pred = prediction(predMat.toarray(), k)
test_ndcg, dev_ndcg = evaluationGlobal(pred, test_set, dev_set, k)
print("test score:",test_ndcg,"\n dev score:",dev_ndcg)
```

UWPop

```
def uwPopMat(df_train, n_items, recency=recency):
  '''
    Calculate the user popularity matrix with the given recency window
  '''
  n_users = df_train.UID.unique().shape[0]
  if (recency>0):
    # Get the number of user baskets Bu
    BUCount = df_train.groupby(['UID'])['order'].max().reset_index(name='Bu')
    # Calculate the denominator which equal to Min(recency,Bu) for each user
    BUCount['denominator'] = np.minimum(BUCount['Bu'],recency)
    # Calculater the order index, form where we start counting item appearance in
        recent orders
    BUCount['startindex'] = np.maximum(BUCount['Bu']-recency,0)
    # Calcualte item appearance in recent orders
    tmp = pd.merge(BUCount, df_train,on='UID')[['UID','PID','order','startindex']]
    tmp =
        tmp.loc[(tmp['order']>=tmp['startindex'])==True].groupby(['UID','PID'])['order'].count().rese
```

```
    tmp = pd.merge(BUCount[['UID','denominator']],tmp,on='UID')
    # finally calculate the recency aware user-wise popularity
    tmp['Score'] = tmp['numerator']/tmp['denominator']
  else :
    # Calculate user-wise popularity for each item
    BUCount = df_train.groupby(['UID'])['order'].max().reset_index(name='Bu')
    BUICount = df_train.groupby(['UID','PID'])['BID'].count().reset_index(name='Bui')
    tmp = pd.merge(BUICount, BUCount, on='UID')
    del BUICount
    tmp['Score'] = tmp['Bui']/tmp['Bu']
    del BUCount
    # get the 3 columns needed to construct our user-wise Popularity matrix
  df_UWpop = tmp[['UID','PID','Score']]
  del tmp
  # Generate user-wise popularity matrix in COOrdinate format
  UWP_mat = sparse.coo_matrix((df_UWpop.Score.values, (df_UWpop.UID.values,
      df_UWpop.PID.values)), shape=(n_users,n_items))
  del df_UWpop
  return sparse.csr_matrix(UWP_mat)
```

IP-CF:

```
def ipcf(df_train, UWP_sparse, n_items,alpha, q, k):
  # Construct the item-basket sparse matrix
  idMax_basket = df_train.BID.max()+1
  item_basket_mat = sparse.coo_matrix((np.ones((df_train.shape[0]),dtype=int),
      (df_train.PID.values, df_train.BID.values)), shape=(n_items,idMax_basket))
  # Convert it to Compressed Sparse Row format to exploit its efficiency in
      arithmetic operations
  sparse_mat = sparse.csr_matrix(item_basket_mat)
  # Caculate the Asymetric Cosine Similarity matrix
  itemSimMat = sim.asymmetric_cosine(sparse_mat, None, alpha, k)
  # recommend k items to users
  UWP_sparse.shape, itemSimMat.shape
  user_recommendations = sim.dot_product(UWP_sparse, itemSimMat.power(q), k)
  return user_recommendations
```

```
n_items = df_products['PID'].unique().shape[0]
UWP_mat = uwPopMat(df_train, n_items, recency=recency)



# prediction
score, pred = prediction(user_recommendations.toarray(), k)
del user_recommendations
# Evaluation
test_ndcg, dev_ndcg = evaluation(score, pred, test_set, dev_set, k)
print("test score:", dev_ndcg)
del score,pred
```

UP-CF:

```
def upcf(df_train, UWP_sparse, n_items, alpha = alpha ,q=q, k=k):
  n_users = df_train['UID'].unique().shape[0]
  df_user_item =
      df_train.groupby(['UID','PID']).size().reset_index(name="bool")[['UID','PID']]
  # Generate the User_item matrix using the parse matrix COOrdinate format.
  userItem_mat = sparse.coo_matrix((np.ones((df_user_item.shape[0])),
      (df_user_item.UID.values, df_user_item.PID.values)), shape=(n_users,n_items))
  # Calculate the asymmetric similarity cosine matrix
  userSim = sim.asymmetric_cosine(sparse.csr_matrix(userItem_mat), alpha=alpha, k=k)
  # recommend k items to users
  user_recommendations = sim.dot_product(userSim.power(q), UWP_sparse, k=k) #r^u_i in
      the paper
  return user_recommendations
```

Extension with product type

New data processing:

```
def instacartExtendProcessing(dataPath, item_threshold=10, basket_threshold=2,
    subdata=subdata):
  '''
  IN:
    dataPath : os path to instacart data
    item_threshold : (default = 10 basket)
```

```
  basket_threshold : (default = 2 basket)
  subdata : (default: 5% of data)
  verbose: Boolean, (default:True)
OUT:
  df_train : DataFrame where columns = [BID, UID, order,articles]
  dev_set, test_set : user-baskets items dict like {UID -> [PID,...], UID ->
      [PID,...], ...}
  df_products: DataFrame where columns = [PID, description, department, category]
'''
# Read products.csv
# Read products.csv
df_products = pd.read_csv(os.path.join(dataPath,"products.csv"))
df_products.columns = ['PID', 'description', 'categoryID', 'departmentID'] #name
    the columns
# Read departments.csv and merge
tmp = pd.read_csv(os.path.join(dataPath,"departments.csv"))
tmp.columns = ['departmentID', 'department']
df_products = pd.merge(df_products, tmp, on='departmentID')
tmp2 = pd.read_csv(os.path.join(dataPath,'aisles.csv'))
tmp2.columns = ['categoryID', 'category']
df_products = pd.merge(df_products, tmp2, on='categoryID')
del tmp, tmp2
# preprocessing

df_order_products_prior =
    pd.read_csv(os.path.join(dataPath,"order_products__prior.csv"))
df_order_products_train =
    pd.read_csv(os.path.join(dataPath,"order_products__train.csv"))
df_order_products = pd.concat([df_order_products_prior,
    df_order_products_train])[['order_id', 'product_id']]
df_order_products.columns= ['BID','PID']
del df_order_products_prior, df_order_products_train,

# Removing all items that appears in less than item_threshold baskets
products_count = df_order_products['PID'].value_counts()
df_order_products=
    df_order_products.loc[df_order_products['PID'].isin(products_count[products_count
    >= item_threshold].index)]
del products_count
```

```
df_orders = pd.read_csv(os.path.join(dataPath,"orders.csv"))[['order_id',
    'user_id', 'order_number', 'eval_set']]
df_orders.columns = ['BID','UID','order', 'set']

# User filtring
# Removing users with less than basket_threshold baskets
user_count = df_orders['UID'].value_counts()
user_filter = user_count[(user_count>=basket_threshold) &
    (np.random.rand(len(user_count))< subdata)]
del user_count
df_orders = df_orders[df_orders['UID'].isin(user_filter.index)]
del user_filter



user_dict = dict(zip(df_orders['UID'].unique(),
    range(len(df_orders['UID'].unique()))))
df_orders['UID'] = df_orders['UID'].map(user_dict)
del user_dict

  # reset product index
df_order_products =
    df_order_products.loc[df_order_products['BID'].isin(df_orders['BID'].unique())]
df_products =
    df_products[df_products['PID'].isin(df_order_products['PID'].unique())]
product_dict =
    dict(zip(df_order_products['PID'].unique(),range(len(df_order_products['PID'].unique()))))
df_products['PID'] = df_products['PID'].map(product_dict)
df_order_products['PID'] = df_order_products['PID'].map(product_dict)
del product_dict

  # Join Tables
df_data = pd.merge(df_orders, df_order_products, on= 'BID')
del df_order_products

  # Setting last baskets as dev/test sets
last_basket_indexes =
    df_data.iloc[df_data.groupby(['UID'])['order'].idxmax()]['BID'].values
df_data.loc[df_data['BID'].isin(last_basket_indexes),'set']='test'
```

```
  df_data.loc[df_data['set']=='prior', 'set'] = 'train'
  del last_basket_indexes

  # train test split data
  df_split =
      df_data[df_data['set']=='test'].groupby(by=['UID'])['PID'].apply(list).reset_index(name='artic
  msk = (np.random.rand(len(df_split))<0.5)
  df_dev, df_test = df_split[msk], df_split[~msk]
  del df_split

  df_train = df_data[df_data['set']=='train'][['UID','BID','order','PID']]

  dev_set = dict(zip(df_dev['UID'],df_dev['articles']))
  test_set = dict(zip(df_test['UID'],df_test['articles']))

  del msk, df_dev, df_test

  return df_train, dev_set, test_set, df_products, df_data
```

DepartmentID one-hot coded matrix

```
df_OneHot = pd.get_dummies(df_products['departmentID'])
print('The shape of the one-hot encoded matrix is : ', df_OneHot.shape)
df_OneHot

beta=5

array = df_OneHot.to_numpy()
arrayT = np.transpose(array)
finalArray = np.dot(array, arrayT)
new = np.array([1,beta])[finalArray]
csr_sparse_OneHot = sparse.csr_matrix(new)
new_matrix_test = csr_sparse_OneHot.multiply(csr_sparse_OneHot.T)
```

Testing

```
n_items = df_products['PID'].unique().shape[0]
UWP_mat = uwPopMat(df_train, n_items, recency=recency)

idMax_basket = df_train.BID.max()+1
```

```
item_basket_mat = sparse.coo_matrix((np.ones((df_train.shape[0]),dtype=int),
    (df_train.PID.values, df_train.BID.values)), shape=(n_items,idMax_basket))
# Convert it to Compressed Sparse Row format to exploit its efficiency in arithmetic
    operations
sparse_mat = sparse.csr_matrix(item_basket_mat)
# Caculate the Asymetric Cosine Similarity matrix
itemSimMat = sim.asymmetric_cosine(sparse_mat, None, alpha, k)

q=1
alpha = 1
k=5

user_recommendations = sim.dot_product(UWP_mat, itemSimMat_enhance.power(q), k)

# prediction
score, pred = prediction(user_recommendations.toarray(), k)
del user_recommendations
# Evaluation
test_ndcg, dev_ndcg = evaluation(score, pred, test_set, dev_set, k)
print("test score:", dev_ndcg)
del score,pred
```