Product Recommendation Engine for Paychex

Names ….
Group 3, DSC 483
Goergen Institute for Data Science
University of Rochester

1. Introduction

With the development of information technology and the popularity of the Internet, people are gradually moving from a lack of information to an era of information overload. According to [1], since the birth of the World Wide Web in 1991, there are over 1.7 billion websites all around the world. In the last ten and twenty years, the number of websites has increased by 629% and 54526%, respectively. And the number is still increasing. Obviously, this has far exceeded the amount of information that people can understand. It brings people to "the paradox of choice", that is "too much choice will increase people's anxiety". [2] In the era of information overload, both customers and producers are facing great challenges. As consumers, it is very difficult to quickly acquire the information (or products/services) they are interested in from the huge amount of information. As producers, how to make the information (or products/services) stand out is also very challenging. Therefore, the recommendation system was born to solve the problem of information overload by recommending the right information/products/services to the right people. In fact, the recommendation system has been used in all aspects of our lives, including recommendations from websites such as e-commerce, movies, music, social networks, etc. And it has also been proven to be of great value in business. For example, 35% of Amazon.com's revenue is generated by its recommendation system [3], and 75% of videos watched on Netflix come from their recommendation system [4].

There are mainly three types of recommendation systems. The first is the content-based recommendation [5]. It assumes that users are interested in the same type of products and will recommend similar content/products/services to users based on their browsing/purchasing history. The second is collaborative filtering (CF) including the user-based CF and item-based CF [6]. This method will find like-minded people for you and recommend information/products/services they liked or bought to you. And the last is hybrid recommendation systems, which incorporate different methods [7]. In addition, there are other methods including association rules-based recommendation [8], social recommendation [9], and demographic-based recommendation [10]. In recent years, with the development and improvement of deep learning technology, multiple recommendation systems combining CNN/RNN/GCN have also appeared [11, 12].

Our sponsor Paychex, as a leading provider of integrated payroll, retirement, and human resource solutions for small to mid-sized businesses, has served over 650,000 payroll clients across 100 locations and pays one out of every 12 American private sector employees with hundreds of solution services [13]. Obviously, it is impractical to customize a solution for each client. Therefore, a recommendation engine is needed to recommend personalized products for clients based on their information and purchasing history, which can eventually improve product sales.

2. Dataset

2.1 Dataset Description

Our dataset is from our sponsor, Paychex, including the information of 382,252 clients and their purchasing history. Table.1 shows the first 5 clients' information.

- ★ The first column (in the left red box) is the client ID, which is a unique number to identify the client.
- ★ Column 2-41 (in the middle red box) are clients' purchasing history, including 40 low-level products (A1A9, B1-B6, C1-C7, D1-D3, E1-E13, F1-F2), which belong to 6 high-level products (A, B, C, D, E, F), and the corresponding relationship is shown in Fig. 1. 1 means the client already owns the product, while 0 means not.
- ★ The last 3 columns (in the right red box) are clients' information. 'Rep.Level' refers to the customer representative level of Paychex with 6 levels (level 1 - level 5, and unknown), which does not represent their ability or seniority. Instead, it depends on the 'Size' and 'Industry' of clients. 'Size' refers to the number of employees in the client's company. And there are 5 levels, from Size 1 to 5 ascendingly. 'Industry' refers to the industry of clients' company, which consists of 13 industries (Industry 1 – Industry 13).

Table I Paychex Recommendation Dataset (First 5 Rows)

	Client_Id	A1	A2	A3	A4	A5	A6	A7	A8	A9	...	E5	E6	E7	E8	E9	F1	F2	Rep.Level	Size	Industry
0	1	0	0	0	1	1	1	0	0	0	...	0	0	0	0	0	0	0	Level 4	Size 3	Industry 10
1	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	Level 5	Size 4	Industry 6
2	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	Level 5	Size 4	Industry 10
3	4	0	0	0	1	1	1	0	0	0	...	0	0	0	0	0	0	0	Level 4	Size 2	Industry 12
4	5	0	0	0	1	1	0	0	0	0	...	0	0	0	0	0	0	0	Level 4	Size 3	Industry 10

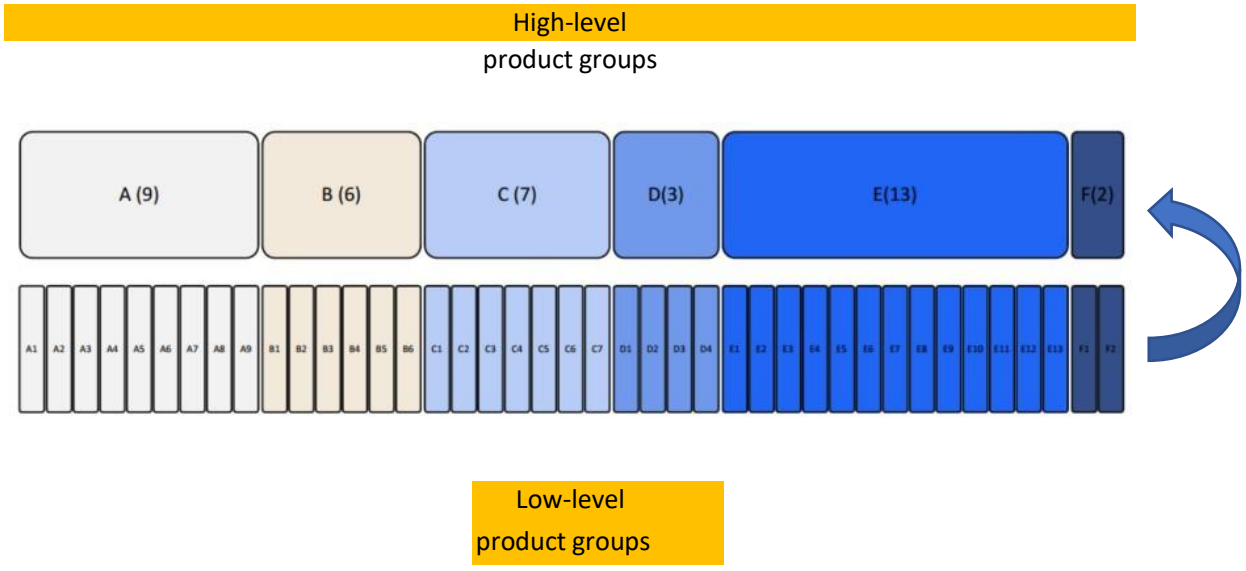


Fig. 1 High-level Product and Low-level Product of Paychex Recommendation Dataset

2.2 Data Preprocessing

Fortunately, in all the data, only one row has an error (D2 F2=2). After consulting with our sponsor, we changed it to 1. No duplicate data and missing values were found in this data set.

3. Exploratory Data Analysis

Before constructing the recommendation system, we need to have a deeper understanding of the data. In this section, we will analyze from four aspects, including the clients' proportion in each demographic category, the product purchase rate, clients' proportion of the number of high-level products purchased, and the unique purchasing history.

3.1 Clients' Proportion in each demographic Category

From the previous section, we know that there are three kinds of demographic information of clients, including 'Rep.Level', 'Size' and 'Industry'. The number of clients in each demographic category are shown in Fig.2 (a), (b) and (c), respectively.

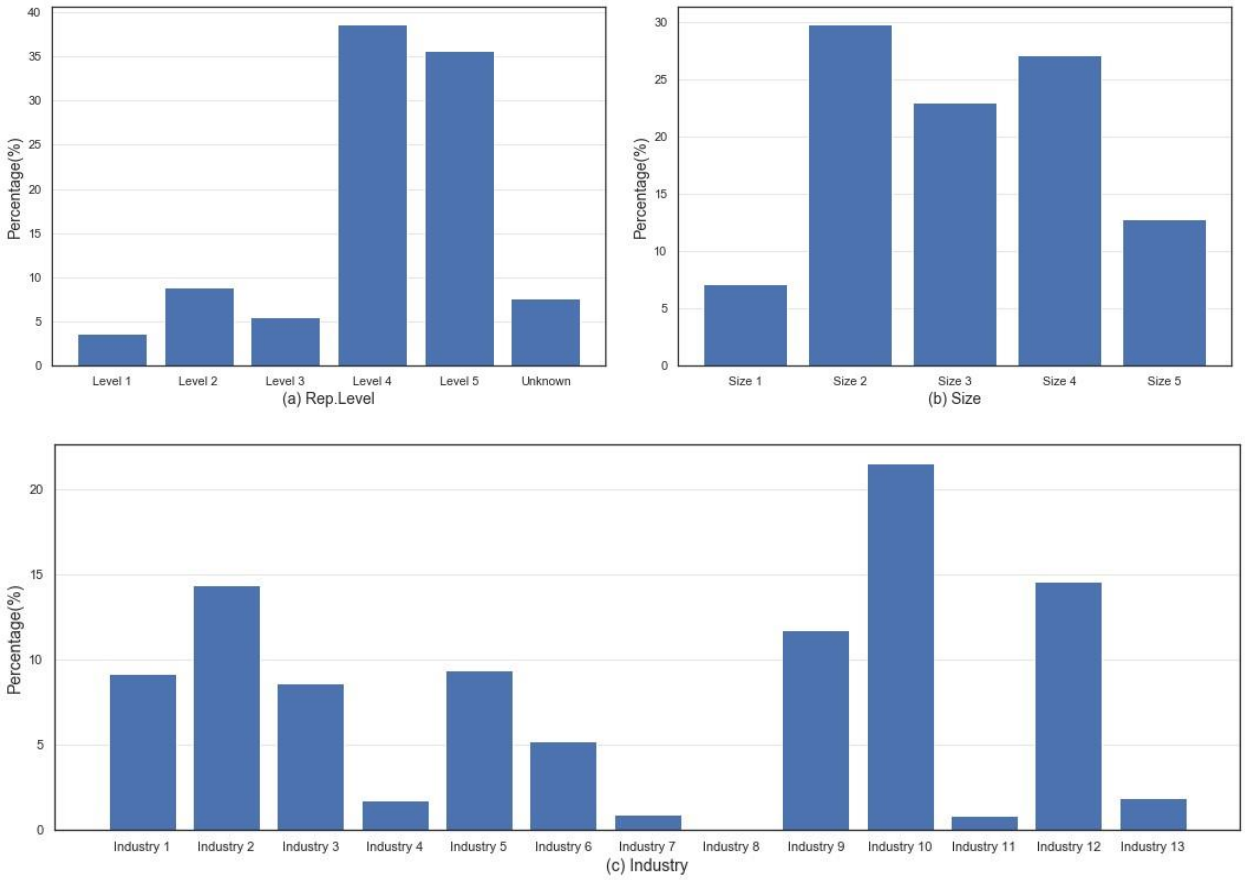


Fig. 2 Clients proportion in each demographic category: (a) Rep.Level, (b) Size, (c) Industry

Fig. 2(a) shows the clients proportion in each 'Rep.Level'. Obviously, the level 4 and 5 have the most clients

(74.25% in total). Interestingly, the 'Unknown' also takes about 7%. Fig.2 (b) illustrates the clients' proportion in each 'Size'. Most clients are medium-sized companies (size 2 to 4, about 80%). Fig. 2(c) demonstrates the distribution of clients' proportion in each 'Industry'. Clearly, the distribution is very uneven. Industry 10 has the most clients (21.55%), while Industry 8 only has 17 clients (0.004%).

3.2 Product Purchase Rate

The purchase rate of low-level products and high-level products are shown in Fig. 3. From Fig. 3 (a), we can see that the distribution of the purchase rate of low-level products is very imbalanced. The highest F1 has 23.47%, while the lowest C7 only has 0.000523%. Even within the same high-level, there is a large gap in purchase rates. Fig. 3 (b) illustrates the purchase rate of high-level products, that is, if any low-level product within the highlevel is purchased, the high-level product is considered to be purchased. As shown in Fig. 3(b), F is the highest one with about 40%, while B is the lowest one with only less than 15%.

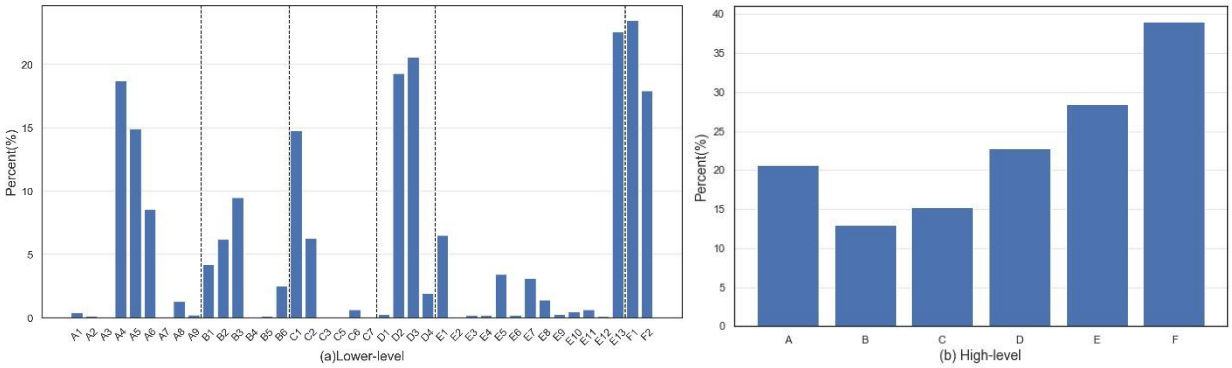


Fig. 3 Purchase rate of Low-level products (a) and High-level products (b)

3.3 Clients' Proportion of the number of High-level Products Purchased

The clients' proportion of the number of high-level products purchased is shown in Fig. 4. Obviously, there are over 25% of clients (105,463 clients) are new clients (who have not bought any product from Paychex), which involves the cold start problem and will be discussed later. Among the remaining clients, the proportion of clients decreases as the number of high-level products purchased increases. Interestingly, the percentage drop almost follows the power-law distribution.

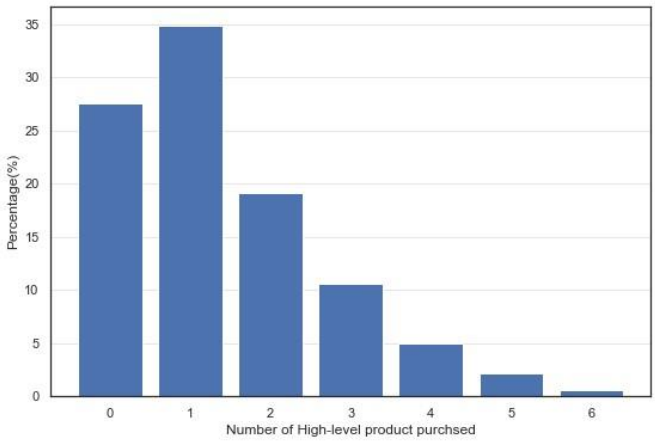


Fig. 4 Percentage Distribution of the number of High-level Products Purchased

3.4 Unique Purchasing History

Although there are no duplicated clients, there are many identical purchasing histories. Among all the 382,525 clients, excluding the 105,463 new clients, the remaining 277,062 clients only have 13257 unique purchasing history, which means many clients have the same purchasing histories, with 20.90 clients per purchasing history on average. And the client purchasing history filtering funnel is shown in Fig. 5(a). In Fig. 5(b) shows the distribution of clients' proportion of purchasing history (proportion of purchasing history in descending order). Clearly, the distribution is still very uneven. The Top 16 purchasing history (PH 1- PH 16, all over 1%) takes over 50% clients. And top 1 PH accounts for 14.57%. That is to say, 40380 clients have the identical purchasing history. By knowing that, we can greatly reduce the calculation load in the following model section.

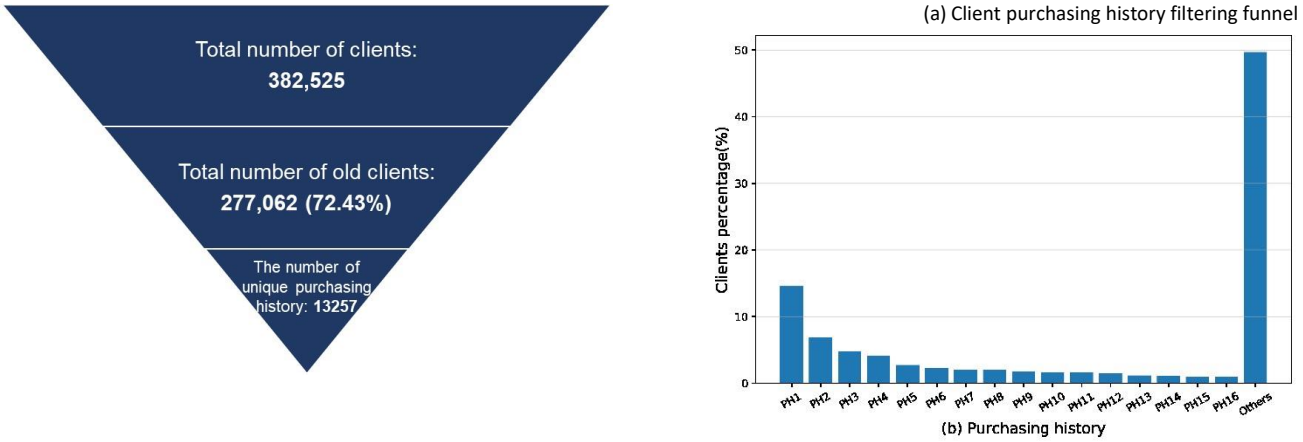


Fig. 5 Clients' purchasing history filtering funnel and distribution (in descending order)

3.5 Summary

In this section, we further analyze the data from four aspects and obtain four findings accordingly. First, most clients are medium-sized companies, and served by Rep.Level 4 and 5. And the Industry distribution is very uneven. Secondly, the purchasing rate in low-level products is very imbalanced, even within the same high-level product. And High-level product F has the highest purchasing rate with around 40%. Then we find there are over 25% new clients in this data set, and the number of clients decreases as the number of high-level products increases. Finally, the fact that many clients have the same purchasing history is found, and the most popular purchasing history owns 40380 clients (with 14.57% among all the old clients).

4. Model Development

Since we want to recommend new products to our clients, the natural way to achieve this goal is to build a recommendation system. According to Wikipedia, recommendation system is one class of model that seeks to predict "rating" or "preference" a user would give to an item [14]. To setup recommendation system, we first need to prepare our dataset into the format called user-item matrix. In such format, each row will represent a user (i.e., client) and each column will represent an item (i.e., low-level product), each entry will be the rating of item from the corresponding user. In our dataset, the numerical rating is substituted by binary score: 1 will represent

the corresponding user have bought this low-level product, 0 will represent the opposite. Fortunately, our dataset has satisfied this criterion (by excluding the 'Rep.Level', 'Size' and 'Industry').

As we mentioned before, there are mainly three types of recommendation systems: content-based filtering, collaborative filtering and hybrid recommendation (not discussed here). The former assumes access to side information about product (e.g., properties of a product), the later does not assume access to side information about product (e.g., does not need to know about product properties) [15]. In this project, since we have no side information about our low-level products, we choose the collaborative filtering approach for modeling.

In the collaborative filtering approach, we can further identify two primary methods: neighborhood methods and latent factor methods [16]. Neighborhood methods will utilize the neighbors (defined by similarities between clients/products) and recommend products that those neighbors purchased. Latent factor methods assume that both products and users live in some low-dimensional space describing their properties and recommend a product based on its proximity to the client in the latent space [16].

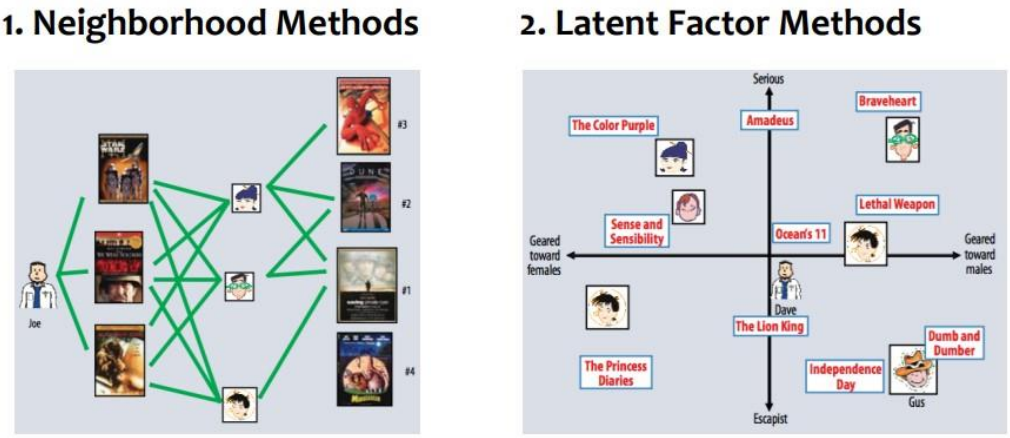


Fig. 6 Two Types of Collaborative Filtering [15]

In this project, we have explored both the neighborhood methods and latent factor methods. Meanwhile, we also applied several tricks to improve model performance.

4.1 Modeling Setup

Before we detail our modeling part, we first finished several setups for later analysis.

4.1.1 Data Cleaning

As we can see from figure 5, among 382,525 clients, there are 105,463 new clients (i.e., clients with no purchasing history). Since we want to establish a recommendation system based on the purchasing history of clients, these new clients will be excluded temporarily for modeling part (regarding the recommendation for these new clients, see 4.6 Cold-Start Problem). Furthermore, we will also exclude demographic features ('Rep.Level', 'Size', 'Industry') when building our collaborative filtering models, these features will be utilized later in the clustering part. After these processing steps, our final dataset will contain 277,062 clients and 40 features (each represents a low-level product). The processed version is the typical user-item matrix as we have mentioned before.

4.1.2 Train-Validate-Test Splitting

Following the usual setting in the machine learning field, we split our dataset into three parts: training, validation and test sets. The training set contains 60% of the data and is used for training models. The validation set contains 20% of the data and is used for tuning hyper-parameters. The test set contains 20% of the data and is used for estimating generalization error.

Later, we will include a clustering technique in our model. To make a fair comparison of model performance, we modify the previous train-validation-test split strategy and utilize the stratified sampling technique. That is, we first execute the train-validation-test procedure in each cluster and then merge the corresponding part in each cluster together as our final train, validation and test set.

4.1.3 Validation / Test Decoding Process

Unlike the usual supervised learning problem, we have no further information about entries with value 0 (i.e., low-level products which client hasn't bought). That means we have no ground truth for model prediction. To compare the performance of different models, we come up with a strategy called 'decoding process'. Specifically, for each client in the validation/test set, we will 'erase' all the products he/she has bought, one at a time, to generate testing samples. Now suppose a client has bought k items, by doing this, we will get k different test samples for this client. Therefore, after the decoding process, our validation/test set will be augmented several times, which causes the explosion of validation/testing time and memory used. We have optimized our code to accommodate the efficiency requirements (seeing our code for more details). One thing to notice here is that if one client has only bought 1 product before, after the decoding process, this will generate a test sample with all 0 (i.e., new clients). Our solution is to drop such test samples.

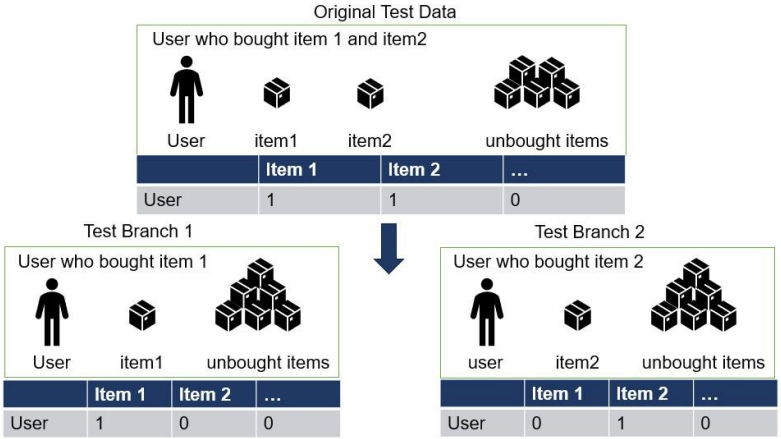


Fig. 7 Decoding Process for Validation/Test Set

4.1.4 Making Prediction

The output of our models for each client will have the same dimension as the input row. We will rank the prediction value and return the product with the highest prediction value.

4.1.5 Evaluation Metric

To estimate the model performance, we have to choose our evaluation metric. In this project, our evaluation metric is recall@1. We prefer recall as the evaluation metric is because that we care the correct prediction more than the wrong prediction, wrong prediction (i.e., advertising clients with uninterested products) is fine in this case but correct prediction (i.e., advertise clients with their desired products) will significantly improve sales. The detailed calculation formula is shown below:

$$Recall@1 = \frac{TP}{TP + FN}$$

Where TP represents True Positive, i.e., the number of products which is 1 is predicted as 1. FN represents False Negative, i.e., the number of products which is 1 is predicted as 0.

4.2 User-based Collaborative Filtering Model

Our baseline model is a user-based collaborative filtering model (i.e., user-based neighborhood method). In this model, we will first calculate the similarities between clients, and then find the k most similar clients to the given user, finally recommend products not bought by this user (illustration can be found in the left part of figure 6).

4.2.1 Similarity Metric

The first question we must deal with is how to measure the similarity between different clients. The similarity measure we choose for this project is cosine similarity. The formula for cosine similarity is shown below, where a is one client, b is another one and k denotes the total number of products. Actually, it calculates the cosine of the angle between two user-vectors. The valid range of cosine similarity range from -1 to 1. One underlying effect of cosine similarity is that it treats the lack of a rating as more similar to disliking the product than liking it [17].

$$cosine\_similarity(a, b) = \frac{a * b}{||a|| * ||b||} \quad \frac{\sum_{i=1}^{k} a_i b_i}{\sqrt{\sum_{i=1}^{k} a_i^2} * \sqrt{\sum_{i=1}^{k} b_i}_2} =$$

4.2.2 Hyper-parameter Tuning

There is one hyper-parameter for our baseline model: the number of neighbors (k). We have experimented with several k values and test their performance in the validation set (seeing Figure 7 for result). The final choice of k is 4000 based on validation recall@1.

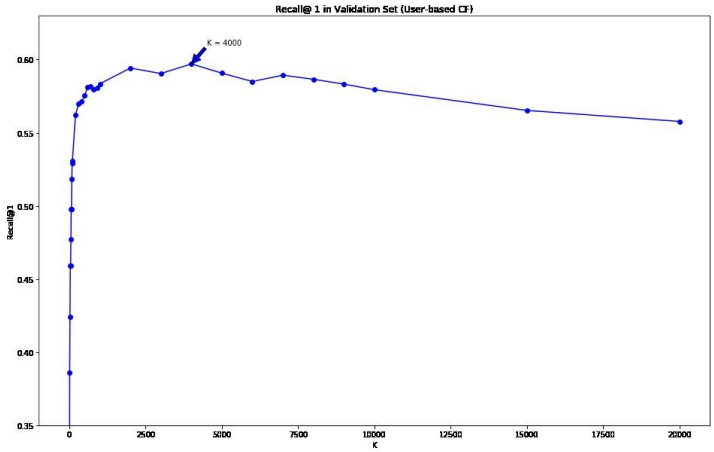


Fig. 8 Recall@1 in Validation Set (User-based Collaborative Filtering)

4.3 Latent Factor Models / Matrix Factorization

Instead of using similarity for prediction, we can also estimate the blank entries in the user-item matrix by conjecturing that the user-item matrix is actually the product of several long, thin matrices. This technique is called latent factor models/matrix factorization and assumes that there is a relatively small set of features of items and users that determine the reaction of most users to most items [17]. Specifically, we have tried two such algorithms: Singular Value Decomposition (SVD) and FunkSVD [14]. We will ignore FunkSVD here because its performance is poor, and we finally drop it.

SVD is a commonly used matrix factorization method. Now supposing our user-item matrix is $X \in R^{m \times n}$ with m clients and n products, then after running SVD, we will get three small matrices: $U \in R^{m \times r}$, $S \in R^{r \times r}$ and $V^T \in R^{r \times n}$. They satisfy the property that $\hat{X} = USV^T \approx X$, i.e., we try to use the multiplication of three small matrices to approximate our original user-item matrix X. U is called left singular matrix, representing the relationship between clients and latent factors. S is a diagonal matrix describing the strength of each latent factor, while $V^T$is a right singular matrix, indicating the similarity between items and latent factors [18]. The reason why we want to decompose our original user-item matrix is that we want to reduce the dimension of data and understand the relationship between user and item in a relatively low-dimension space (latent space).

$$\underset{m \times n}{\hat{X}} \approx \underset{m \times r}{U} \; \underset{r \times r}{S} \; \underset{r \times n}{V^T}$$

Fig. 9 SVD Decomposition [18]

Besides, we can change the number of components (n_component) in diagonal matrix S, to achieve the effect of data compression like PCA. Figure 10 shows the validation recall@1 for different 'n_components'.

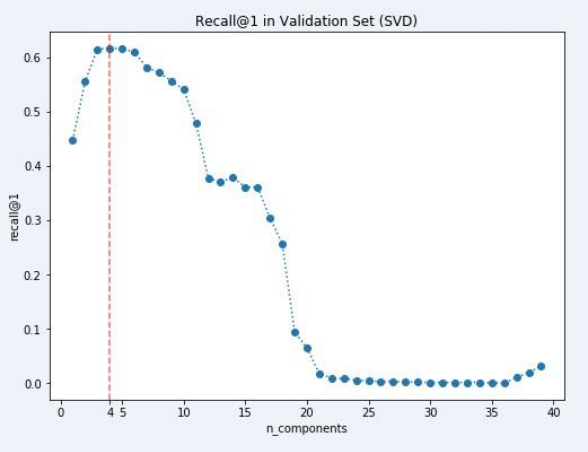


Fig. 10 Recall@1 in Validation Set (SVD)

4.4 Data Imputation

One of the biggest troubles when building recommendation systems is that the user-item matrix is very sparse. That means for most clients, they have bought only a few products, leaving too many 0's in each row. Sparseness usually affects the performance of recommendation systems. For instance, SVD tends to predict the extreme value in this case to minimize the RMSE. In this case, the performance of the model will be as low as zero. To relieve the problem of sparseness, we can impute some low value for each 0. This trick is called data imputation. In this project, we explore two data imputation methods: one is SVD, another is low-value injection.

4.4.1 SVD

SVD is originally used for finding the main components of the data by removing the noise of data, which results in a slight change to values in the matrix. Surprisingly, SVD makes 0 nonzero, and hence can be considered as a way of imputation. We can incorporate this strategy with our baseline model, the corresponding result is shown in Figure 11. We can see that it is better than our baseline model.

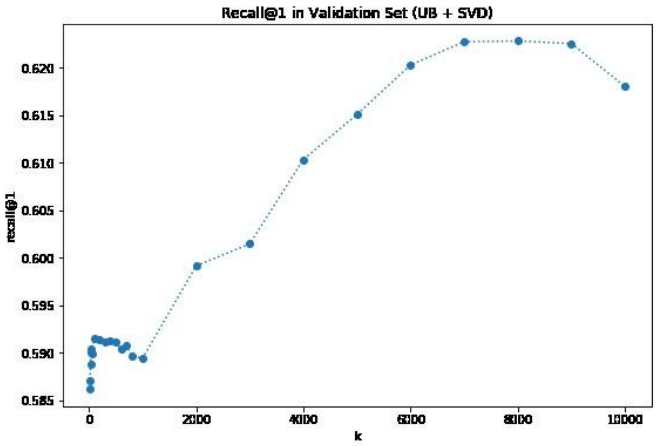


Fig. 11 Recall@1 in Validation Set (UB + SVD)

4.4.2 Low Value Injection

Instead of using SVD, we also came up with another strategy for data imputation called low-value injection.

Specifically, we will impute all the 0 in the dataset with $\frac{\lambda}{\#\ items\ bought}$ for each client. The intuition behind this method is that for those clients who have bought lost of products, their willingness to buy a new product will lower than those with few purchase histories. Therefore, the imputed value for those who have bought many products will lower than those with few purchases.

4.4.3 SVD + Low Value Injection

Imputation can provide more information for the collaborative filtering algorithm, so we think it might also provide more information for the SVD. Hence, we combined two data imputation methods, but from Figure 12 we can see the combined method does not improve the Recall@1 compared to UB+SVD result in Figure 11. However, the low-value injection method improves Recall@1 when it meets with clustering.

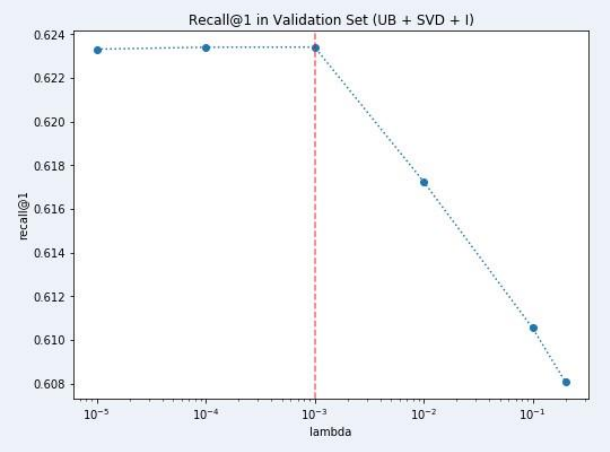


Fig. 12 Recall@1 in Validation Set (UB + SVD + I)

4.5 Clustering

In our baseline model, we haven't utilized the demographic information of clients (i.e., 'Size', 'Industry', 'Rep.Level'). One possible method is using the information to divide these clients into several groups. However, traditional clustering algorithms such as K-Means, K-Prototype cannot work or do not have good performance.

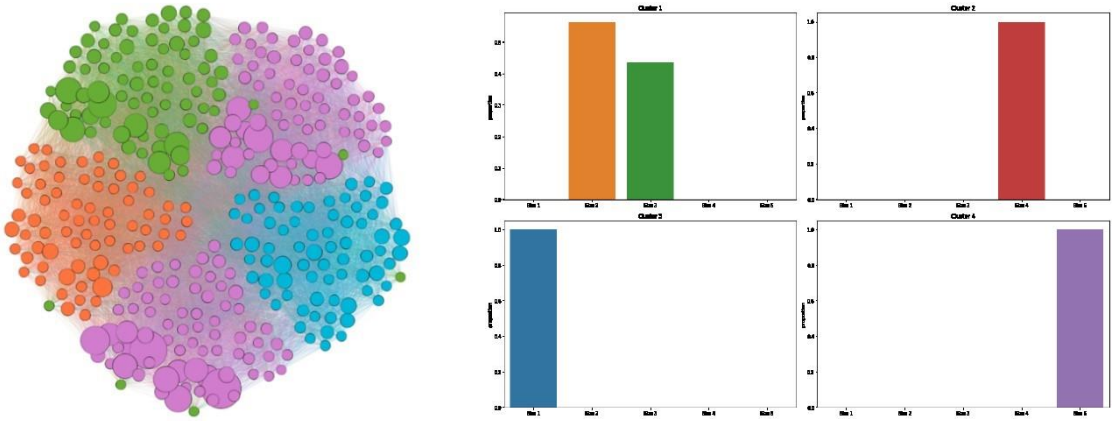
Here, we borrowed the idea from graph theory and complex network to construct an undirected weighted graph. Since many clients have the same demographic information, we can consider that these clients are similar and may have similar requirements. Therefore, we abstract clients with identical demographic information (that is, all 3 kinds of demographic information are the same) into a node, and use edges to connect the nodes with the same demographic information (1 or 2 kinds are the same). The more same demographic information between two nodes, the closer they are. Therefore, we use the reciprocal of the number of the same demographic information between two nodes as the weight of the edge, which is also the reciprocal of the distance between the nodes.

In order to divide the nodes, we utilized the fast Newman algorithm [19], based on the modularity maximization to identify the community structure. It was designed to measure the strength of the division of a network into

modules (also called groups, clusters or communities). Networks with high modularity have dense connections between the nodes within modules but sparse connections between nodes in different modules [20].

After running Newman's fast algorithm, we have detected four clusters (figure 9). We can see the well-separation between different clusters. Later, we further check the driven factors of clustering and find that 'Size' plays a critical role. Cluster 1 contains only the clients with 'Size2' and 'Size3', which represent middle-size companies. Cluster 2 contains only the clients with 'Size4', which represent middle-to-large size companies. Cluster 3 contains only clients with 'Size1', which represent small-size companies. Cluster 4 contains only the clients with 'Size5', which represent large-size companies. Besides, we noticed that 'Rep.Level' and 'Industry' have no effect on finding different clusters.

Once we have grouped our clients into different clusters, we can implement our collaborative filtering methods individually for each cluster, which would improve model performance potentially. From Figure 14, we can see that clustering does help to improve the performance.



(a) Clusters (b) 'Size' in each cluster

Fig. 13 Detected Clusters (a) and Distribution of 'Size' for each cluster (b)

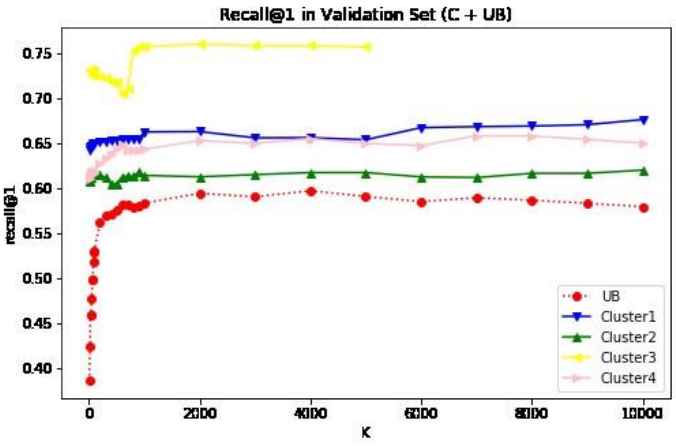


Fig. 14 Recall@1 in Validation Set (C + UB)

4.6 Cold Start Problem

Now we have the recommendation engine for normal clients (i.e., clients with at least one product), we still need to find a way to make a recommendation for new clients (i.e., clients with no purchasing history). This problem is usually called the cold start problem, the performance of collaborative filtering models will degenerate a lot in this case. We bypassed this problem before by ignoring all the new clients, but we still need to give recommendations for these clients. Our solution is by modeling the joint probability distribution in the highlevel. Specifically, we first need to establish the joint probability as follows:

$$P(A, B, C, D, E, F, rep\_level, industry, size)$$

Here we have converted low-level features into high-level features. The way we build joint probability distribution is straightforward: we will estimate each probability combination by its frequency. The Weak Law of Large Numbers makes sure that frequency will converge to probability when we have enough samples. In reality, there are only we have total 24960 unique combinations

To make a prediction for new clients, we can utilize the sum rule and product rule of probability [21]. For example, the probability of buying A given the 'Rep.Level', 'Industry' and 'Size' are:

$$P(A = 1 | rep\_level, industry, size) = \sum_{B}\sum_{C}\sum_{D}\sum_{E}\sum_{F} P(A = 1, B, C, D, E, F, rep\_level, industry, size)$$

Similarly, we can calculate corresponding probabilities for B, C, …, F. The final prediction will base on the ranking of these conditional probabilities.

5. Performance and results

After choosing the best hyper-parameters for each model, we evaluated the test performance for each model. Here we have used some abbreviations for some models. 'UB' denotes the user-based collaborative filtering model. 'SVD' denotes the SVD for imputation, 'I' denotes the low-value injection, 'C' denotes the clustering.

Table II Test Performance for Different Models

Models	Recall@1 on Test for best hyper-parameters
UB	59.71%
UB+SVD	62.28%
UB+SVD+I	62.34%
UB+SVD+I+C	65.31%

Table 2 reports the accuracies of best hyper-parameters for every model. It is found that the UB+SVD+I+C has the highest Recall@1. SVD improves the performance of UB, which indicates removing the noise of the data and a little bit randomness has a positive effect on Recall@1. However, low-value injection improves UB+SVD by a tiny percentage, but we do keep it because compared to UB+SVD+C (Best Recall@1 63.23%, not listed above), the injection method improves a lot.

6. Conclusion & Future works

In this project, after simple preprocessing of the Paychex dataset, we first conducted an exploratory data analysis, and found that there are a lot of clients who have the same purchasing histories. Then we built four recommendation systems based on the clients' purchase history and their categorical information (Rep.Level, Size, Industry). And the last model (User-based collaborative filtering + SVD + Imputation + Graph clustering) has the best performance, with 65.31% recall at 1. Finally, after excluding the purchased high-level product(s), the score of the highest low-level product in each high-level product is extracted and sorted in descending order to obtain the final recommendation results.

There are still a lot of potential improvements. For example, we did not find a good way to handle the weights of duplicated purchase history. And the model evaluation metrics can be extended such as recall at k. In addition, other similarity measurements can also be used, such as the Pearson correlation similarity and Jaccard similarity.

Reference

[1] https://www.internetlivestats.com/total-number-of-websites/

[2] https://en.wikipedia.org/wiki/The_Paradox_of_Choice

[3] http://rejoiner.com/resources/amazon-recommendations-secret-selling-online/

[4] https://medium.com/netflix-techblog/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429 [5] Pazzani, M. J., & Billsus, D. (2007). Content-based recommendation systems. In The adaptive web (pp. 325-341). Springer, Berlin, Heidelberg.

[6] Sarwar, B. M., Karypis, G., Konstan, J. A., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. Www, 1, 285-295.

[7] Bostandjiev, S., O'Donovan, J., & Höllerer, T. (2012, September). TasteWeights: a visual interactive hybrid recommender system. In Proceedings of the sixth ACM conference on Recommender systems (pp. 35-42). ACM.

[8] Lin, W., Alvarez, S. A., & Ruiz, C. (2000). Collaborative recommendation via adaptive association rule mining. Data Mining and Knowledge Discovery, 6, 83-105.

[9] Tang, J., Hu, X., & Liu, H. (2013). Social recommendation: a review. Social Network Analysis and Mining, 3(4), 1113-1133.

[10] Zhao, X. W., Guo, Y., He, Y., Jiang, H., Wu, Y., & Li, X. (2014, August). We know what you want to buy: a demographic-based system for product recommendation on microblogs. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 1935-1944). ACM.

[11] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., & Leskovec, J. (2018, July). Graph convolutional neural networks for web-scale recommender systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 974-983). ACM.

[12] Hidasi, B., Karatzoglou, A., Baltrunas, L., & Tikk, D. (2015). Session-based recommendations with recurrent neural networks. arXiv preprint arXiv:1511.06939.

[13] https://www.paychex.com/corporate

[14] https://sifter.org/~simon/journal/20061211.html
[15] https://www.cs.cmu.edu/~mgormley/courses/10601b-f16/lectureSlides/lecture26-mf.pdf
[16] Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. Computer, (8), 30-37.
[17] http://infolab.stanford.edu/~ullman/mmds/ch9.pdf
[18] https://hackernoon.com/introduction-to-recommender-system-part-1-collaborative-filtering-singularvalue-decomposition-44c9659c5e75
[19] Newman, M. E. (2004). Fast algorithm for detecting community structure in networks. Physical review E, 69(6), 066133.
[20] https://en.wikipedia.org/wiki/Modularity_(networks)
[21] Bishop, C. M. (2006). Pattern recognition and machine learning. Springer.