Team 07 - Update 2

CS6220 - Data Mining Techniques - Fall 2017
Northeastern University

Nakul Camasamudram, Rosy Parmar, Rahul Verma, Guiheng Zhou

November 22, 2017

1 Introduction

We are building a recommender system that provides personalized recommendations based on prior implicit feedback using "The Instacart Online Grocery Shopping Dataset 2017". Instacart is an American company that operates as a same-day grocery delivery service. This anonymized dataset contains a sample of over 3 million grocery orders from more than 200,000 Instacart users.

1.1 Data Analysis

Orders from Instacart are available in four .csv files: "orders.csv", "order_products__train.csv", "order_products__prior.csv" and "sample_submission.csv". The key to understanding the dataset and the train/test split is the orders table ("orders.csv").

Take for example User 1[Fig 1] , who happens to be a train user. User 1 has 10 prior orders, and 1 train order whose details are provided in "order_products__prior.csv" and in "order_products__train.csv" respectively.

Similarly, User 4 is a test user. He has 5 prior orders, and his 6th is a test order. Their details are available in "order_products__prior.csv" and "sample_submission.csv" respectively.

order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order
2539329	1	prior	1	2	08	
2398795	1	prior	2	3	07	15.0
473747	1	prior	3	3	12	21.0
2254736	1	prior	4	4	07	29.0
431534	1	prior	5	4	15	28.0
3367565	1	prior	6	2	07	19.0
550135	1	prior	7	1	09	20.0
3108588	1	prior	8	1	14	14.0
2295261	1	prior	9	1	16	0.0
2550362	1	prior	10	4	08	30.0
1187899	1	train	11	4	08	14.0

(a) User 1

3343014	4	prior	1	6	11	
2030307	4	prior	2	4	11	19.0
691089	4	prior	3	4	15	21.0
94891	4	prior	4	5	13	15.0
2557754	4	prior	5	5	13	0.0
329954	4	test	6	3	12	30.0

(b) User 4

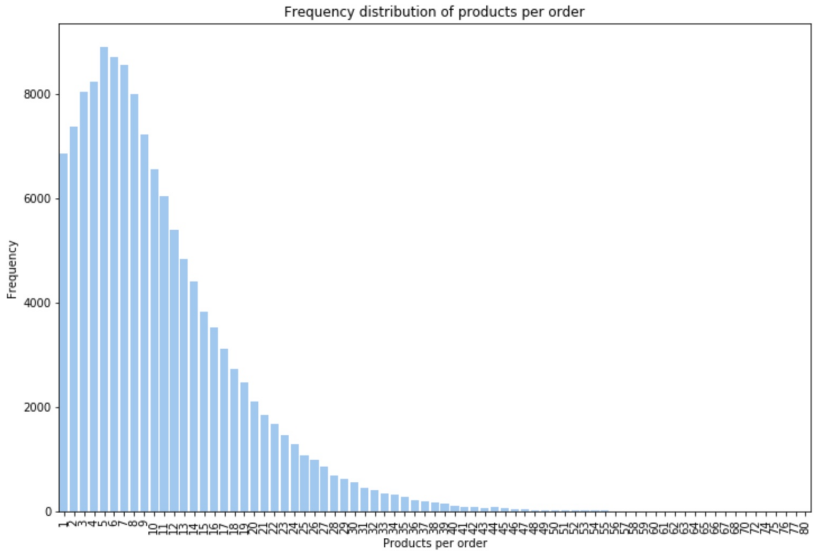
Figure 1: Train/Test Split

	order_id	product_id	add_to_cart_order	reordered	product_name	aisle_id	department_id	aisle	department
0	2	33120	1	1	Organic Egg Whites	86	16	eggs	dairy eggs
1	2	28985	2	1	Michigan Organic Kale	83	4	fresh vegetables	produce
2	2	9327	3	0	Garlic Powder	104	13	spices seasonings	pantry
3	2	45918	4	1	Coconut Butter	19	13	oils vinegars	pantry
4	2	30035	5	0	Natural Sweetener	17	13	baking ingredients	pantry

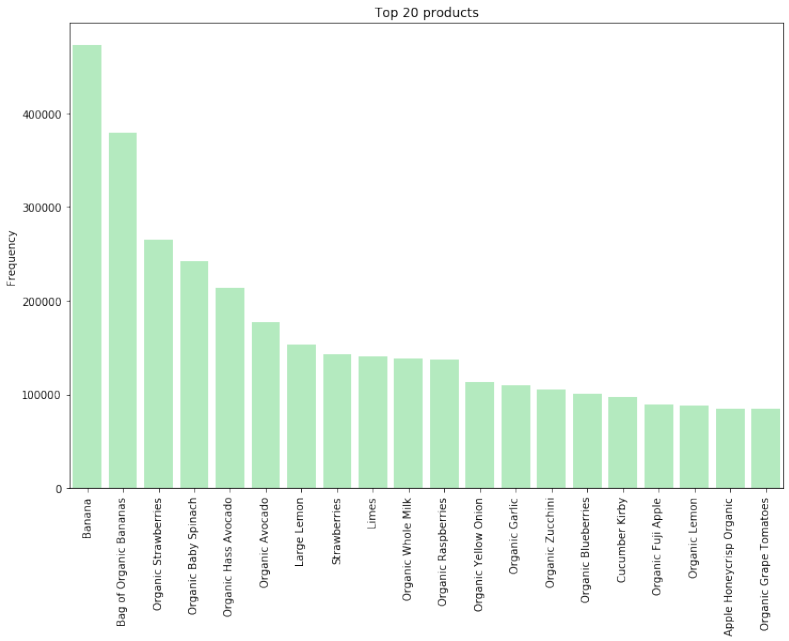
Figure 2: Merged Prior Orders

Figure 2 is a glimpse at "order_products__prior.csv" when merged with three other .csv files that represent products, aisles and departments. The format of "sample_submission.csv" and "order_products__train.csv" is exactly the same.

The below figures depicts the product frequency distribution across orders as well as the most popular products. The "long tail" phenomenon is clearly visible in the former.[1]



[1] A more thorough data analysis was presented in Update 1. It could not be included here due to space constraints



1.2 Challenges

We are using algorithms specifically suited for processing implicit feedback. It is important to highlight unique characteristics of implicit feedback, which prevent the application of algorithms that were designed for explicit feedback data.

1. The dataset has information about users prior purchases from which we can infer what they like. However, there is no concrete metric to deduce what a user dislikes.

2. Just because a user purchased a product, does not necessarily mean he/she likes the product. The purchase could've been made as a gift or perhaps after receiving the product, the user might've been disappointed with it. Also, the frequency of product purchases doesn't necessarily indicate a user's preference, it's more of an indicator for a user's confidence in the product. Hence, a user's true preferences can only be guessed.

3. Systems dealing with explicit feedback are generally evaluated using metrics such as mean average error. With implicit systems, the recommender's output is compared with a user's current purchase on the test set using set based measures.

2 Approaches

2.1 Reviewed Approaches

SVD++

We considered a variant of Singular Value Decomposition(SVD) as an initial matrix factorization. SVD++ is variant of SVD that considers implicit relationships apart from explicit rating information. However, after careful review we decided not to go ahead with it for

a couple of reason. First, it is memory intensive. Secondly, and more importantly, it is an explicit-first approach with additional consideration for implicit data. Since our dataset consists only of past purchases(i.e implicit only), it would not work well with SVD++.

Considering temporal effects
We wanted to consider temporal effects and weigh products according to how recently they were bought. However, the dataset had prior purchases stretching back by only a month and it didn't make sense to weigh purchases in such a short period also considering that we observed monthly and weekly purchases cycles in our exploratory data analysis.[2]

Non-negative Matrix Factorization (NMF)
We considered converting our user-product matrix into an explicit ratings matrix by scaling all the prior product purchase frequencies into a fixed range, and then applying Non-Negative Matrix Factorization [1] to automatically extract sparse and meaningful features. On further research we concluded that this would result in sub-par results. Instead, we could go for a Matrix Factorization approach where, we interpret the values in the user-product matrix as a measure of confidence of a user's preferences.[3]

2.2 Neighborhood-based Methods

2.2.1 Term Frequency-Inverse Document Frequency (TF-IDF)

2.2.1.1 Overview
In this method, we made an analogy:

1. A product a user purchased is considered as a term, and
2. the purchase history of a user (hereinafter also denoted as UPH), comprised of varied products, is treated as a document.

Thus, **Term Frequency** is represented by the number of occurrences of a product in the user purchase history; **Inverse Document Frequency** of a product p is associated with the number of user purchase histories and the number of user purchase histories containing this product, which can be denoted as:

$$idf_p = \log(\frac{\#(UPH)}{\#(UPH\ containing\ p)}).$$

We recommend products based on the similarity among users. Once TF-IDF scores for each user purchase history are created, we can calculate cosine similarity between two users, i and j, based on the TF-IDF scores:

$$sim(i,j) = cos(\vec{i},\vec{j}) = \frac{\vec{i}\cdot\vec{j}}{\|\vec{i}\|_2 * \|\vec{j}\|_2}$$

[2] see Update 1
[3] see Section 2.3.1

2.2.1.2 Concrete Steps

1. Data preprocessing on prior dataset, generating **user-products** and **product-frequency** tables, as shown in Figure 3(a)(b).
2. Calculate the term frequency (TF) table according to user-products table, where each row indicates an instance of user purchase history and each column an instance of product with its occurrences in each user purchase history, as shown in Figure 3(c).
3. Calculate inverse document frequency (IDF) scores for each product, as shown in Figure 3(d).
4. Multiply IDF score vector to each row of TF table, generating TF-IDF Score matrix, as shown in Figure 3(e).
5. Select one user as the target to whom the products will be recommended, calculate the cosine similarity with other users, and pick the top similar ones, as shown in Figure 3(f).
6. Exclude the user (in most cases, the target user) with the same purchase history as the target user, implement **set-difference** on the products purchased by similar users and those by the target user, and select the top 10 popularity (namely overall sales) to recommend.

2.2.1.3 Result Analysis

The following results exhibit the actual products purchased by user with id = 1 and the recommended products for this user.

```
Actual products bought by User 1:
['Organic Honeydew', 'Cold Brew Coffee Tahitian Vanilla', "Spot's Pate Cat Grain Free Ground Whitefish", 'Epic Fruit
& Yogurt Filled Pouches', 'Beet Kombucha', 'Broccoli Squash Carrots Onion Red Pepper Steamables', 'Grape Nut Flakes C
ereal', 'Triple Distilled Irish Whiskey', "Steam'ables Green Peas", 'Cilantro Bunch', 'Peachtree Schnapps', 'Original
Pretzel Crisps', 'Chocolate Caramel Pudding Snack Pack', 'Pasta & Enchilada Sauce, Organic, 7 Veggie', '80% Lean Grou
nd Beef', 'Creamy Chicken & Shrimp in a Parmesan Alfredo Sauce', 'Warrior Blend Vanilla Dietary Supplement', 'Organic
Creamy Cashewmilk']

Recommended products for User 1:
['Brioche Bachelor Loaf', 'Petit Suisse Fruit', 'Premium Genoa', 'Original Antacid & Pain Relief Tablets 36 Ct', 'Max
imum Strength Anti-Itch Creme', 'CleanWear Ultra Thin Regular with Wings Pads', 'Valdosta Pecans With Cranberries, Bl
ack Pepper & Orange Zest', 'Sharpened No. 2 Pencils', 'Traditional Refried Pinto Beans', 'Gluten Free Pecan Shortbrea
d Cookies']
```

After closer look at this user's purchase history, he/she seems to prefer buying foods, especially organic food, snacks, fruits and vegetables. And the recommendation system based on TF-IDF method offers some snacks, fruits, and vegetables. However, the recommendation system doesn't offer organic food; and it also offers some unrelated products like pencils, because currently the generation of recommendations is simply based on set-difference of products purchased by similar users. This will be elaborated in open issues.

2.2.1.4 Open Issues

1. Large dataset results in longer processing time. It takes estimated 50 minutes - 1 hour to calculate term frequency, estimated another half an hour to complete the rest of steps. Can we find a better and faster way to process the data?

2. Processing large dataset also gives rise to the constant kernel-restarting for Jupyter notebook. This is mainly caused by memory shortage during running. Is there a way to expand the memory Jypyter notebook can exploit or it's better to sample the prior data?

3. Set-difference ignores the categories of products, which will lead to missing or unrelated recommendation. The departments or aisles which products belong to should also be considered during recommendation generation.

user_id	products
1	[196, 14084, 12427, 26088, 26405, 196, 10258, ...
2	[32792, 47766, 20574, 12000, 48110, 22474, 165...
3	[9387, 17668, 15143, 16797, 39190, 47766, 2190...
4	[36606, 7350, 35469, 2707, 42329, 7160, 1200, ...
5	[15349, 21413, 48775, 28289, 8518, 11777, 3171...

(a) user-products

product_id	frequency
24852	472565
13176	379450
21137	264683
21903	241921
47209	213584

(b) product-frequency

	1	2	3	4	5	6	7	8	9	10	...	49679	49680	49681	49682	49683	49684	49685	49686	49687	49688
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

(c) Term Frequency

product_id	idf score
1	6.353328
2	8.386503
3	8.498981
4	7.736841
5	11.543503

(d) IDF Scores

	1	2	3	4	5	6	7	8	9	10	...	49679	49680	49681	49682	49683	49684	49685	49686	49687	49688
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.640368	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0

(e) TF-IDF Scores

user_id	similarity
1	1.000000
38487	0.708020
61533	0.560269
63731	0.540606
39853	0.529910

(f) Top Similar Users

Figure 3: Term Frequency - Inverse Document Frequency

2.2.2 Word2Vec Similar Product Based Model

Word2Vec:

Word2Vec uses models that are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Given a list of sentences we can find the similar and dissimilar words to a given Word.
Our Approach is to treat **orders** as sentences and **products** as the **words**

With the help of Gensims Word2Vec[4] we find Similar products to a given product:

1. We make product_id as string first in the table (order_id,product_id).
2. Then we make of list of product_id against a order_id and save it as list(product_id)
3. Then we find the length of longest list(product_id), window = longest list.
4. Since there is no sequence characteristics of the products in an order -Because each product in an order is independent on the orders that were chosen before it in the cart- we should have a training window huge enough to accommodate all the products together or else we imply that products that are far apart even though theyre in the same cart are dissimilar which is not true.
5. We make a Model out of these list(product_id) sentences.
6. For a given product_id model returns list of k most similar product_ids which we leverage to build the Recommender System.

Recommender System:

1. Now we Merge the tables (user_id,order_id) and (order_id,product_id) and find k most frequent product_ids for a given user_id
2. For every product_id in the k most frequent product_ids for a given user_id, we find the most similar product_id returned by the Model

[4]Deep learning with word2vec: https://radimrehurek.com/gensim/models/word2vec.html

	order_id	product_id
0	2	33120
1	2	28985
2	2	9327
3	2	45918
4	2	30035

(a) order-product

	order_id	user_id
0	2539329	1
1	2398795	1
2	473747	1
3	2254736	1
4	431534	1

(b) order-user

	product_name
product_id	
1	Chocolate Sandwich Cookies
2	All-Seasons Salt
3	Robust Golden Unsweetened Oolong Tea
4	Smart Ones Classic Favorites Mini Rigatoni Wit...
5	Green Chile Anytime Sauce

(c) Products

	order_id	quantity
0	2	[33120, 28985, 9327, 45918, 30035, 17794, 4014...
1	3	[33754, 24838, 17704, 21903, 17668, 46667, 174...
2	4	[46842, 26434, 39758, 27761, 10054, 21351, 225...
3	5	[13176, 15005, 47329, 27966, 23909, 48370, 132...
4	6	[40462, 15873, 41897]

(d) order-product-list

```
# taking the longest order length
longest = np.max(df_order_products_list_sample["quantity"].apply(len))
print(longest)
```

```
65
```

```
# Making Orders to sentences of word2Vec
sentences = df_order_products_list_sample["quantity"].values
# print(sentences)
```

```
# Model on WordtoVec for Similar Prodcuts
model= gensim.models.Word2Vec(sentences, size=100, window=longest, min_count=2, workers=4)
```

```
#vocabulary of all the keys:
vocab = list(model.wv.vocab.keys())
```

(e) Making Model

```
# k_freq is k most items a user has purchased and n is the number of
# similar item we want to find corresponding to a product
k_freq=5;n_similar=1;

#  Find  Most Similar item given the ProductID; Exception added if
#  the product not in library
def find_similar(index):
    try:
        if index in vocab:
            return(model.most_similar(positive=[vocab[int(index)]], topn=n_similar))[0][0]
        return(index)
    except IndexError:
        return(index)

# Given a row  of (user_id,Products_list) it finds the k suggestion for
# him by finding the most similar item to his k frequent puchases
def tranform_row(row):
    most_frequnt_k_list=Counter(row['quantity']).most_common(k_freq)
    return_list=[find_similar(item[0]) for item in most_frequnt_k_list]
    return return_list
```

```
# dataFrame to store the Suggestion Product List
df_user_products_list_suggestion=df_user_products_list_sample
```

```
# Finds K best suggestions and stores in the quantity Column
df_user_products_list_suggestion['quantity']=df_user_products_list_sample_counter.apply(lambda row: tranfo
rm_row(row),axis=1)
```

(f) Making the Basic Recommendation System

0	23	[28199, 23106, 9547, 33819, 42959]
1	27	[24852, 13409, 31231, 33754, 11123]
2	66	[32226, 23270, 20738, 31829, 43076]
3	90	[20985, 27582, 7118, 13733, 18352]
4	150	[16953, 36393, 47144, 35518, 33313]

(g) Top 5 Suggestion against every user

Figure 4: Word2Vec

2.3 Latest Factor Methods

2.3.1 Implicit Alternating Least Squares [2]

The key idea of the implicit ALS is to transform a user item matrix of product purchase frequencies to a confidence matrix C_{ui} for users u and items i.

$$C_{ui} = 1 + \alpha R_{ui}$$

α represents a linear scaling of the rating preferences (in our case number of purchases) and R_{ui} is the original matrix of purchases. The paper suggests $\alpha = 40$ to be a good starting point.

Now, similar to other matrix factorization methods, the goal is to find user-factor vectors $x_u \in \mathbb{R}^f$ and item-factor vectors $y_i \in \mathbb{R}^f$ for each user and item. These factors are computed by minimizing the cost function:

$$\min_{x,y} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2)$$

This then leads to an alternating-least-square optimization process, where the algorithm alternates between re-computing user-factors and item-factors with each step guaranteed to lower the value of the cost.

In our case, we are using the library "implicit"[5] to perform ALS optimization.

The following results were obtained by using the model with default parameters.

```
Actual products bought by user: 17
['Grade A Extra Large Eggs', 'Select-A-Size Paper Towels, White, 2 Huge Rolls = 5 Regular Rolls  Towels/Napkins', 'Li
ght Spread Butter Substitute', 'Strawberries', 'Raspberries', 'Ultra Soft Bathroom Tissue Double Rolls']
Recommendations for user: 17

Recommended products bought by user: 17
['Gochujang Sauce', 'Green Chile Anytime Sauce', 'Organic Honeycrisp Apples', 'XXX Acai Blueberry Pomegranate', 'Whea
t Chex Cereal', 'Brioche Bachelor Loaf', 'Yogurt Strawberry Pomegranate', 'Organic Sprouted Barley Bread', 'Nutter Bu
tter Cookie Bites Go-Pak', 'Fabric Softener, Geranium Scent']
```

Though the model is far from perfect, there are interesting similarities in the products that are being recommended and those that were actually purchased. For example, Strawberries and Raspberries were purchased, but, the model recommends 'XXX Acai Blueberry Pomegranate' and 'Yogurt Strawberry Pomegranate'. Also, 'Light Spread Butter Substitute' was purchased, but, 'Organic Sprouted Barley Bread', 'Nutter Butter Cookie Bites Go-Pak' and 'Brioche Bachelor Loaf' was recommended

3 Evaluation

Recommendations for each user are ordered list of products, from the most preferred to the least preferred. The dataset has no feedback about product that are disliked and hence, precisiion based metrics are not appropriate. Instead, recall-oriented measures are applicable since a prior purchase of a product is an indication of liking it.

[5]Fast Python Collaborative Filtering: https://github.com/benfred/implicit

After training the respective models, we are generating recommendations for each user in the test dataset. We will then compute the mean recall over all the users in test dataset by comparing the actual current purchase of the user and the products recommended

$$\text{Mean Recall} = \sum_{\text{test users}} \frac{|\{\text{recommended}\}| \cap |\{\text{actual}\}|}{|\{\text{actual}\}|}$$

Furthermore, we will be evaluating all the models by comparing them to a baseline model that suggests the most popular items to every user.

4 Current Status

1. In TF-IDF method, a TF-IDF score matrix is generated based on the first 100,000 purchase histories of users, and 10 products are recommended according to the products bought by the users most similar to the target one.

2. In the Word2Vec appproach, product-product similarities have already been computed. Using this information, user-user similarity has to be computed based on their prior purchases.

3. At present, Implicit ALS generates recommendations for all users. However, further optimization has to be performed using cross validation to choose the optimal parameters.

5 Next Steps

1. We came across Logistic Matrix Factorization [3] which optimizes the log loss instead of linear loss. It seems to work better for the implicit data in some situations.

2. We will try to build a hybrid model that incorporates features of matrix factorization as well as neighborhood models.

3. For model evaluation, instead of just using Recall, we can somehow consider relative difference between the actual product purchased and our suggestions. For example if purchased product was "Ceramic Jar Medium" and the model's recommendation was "Ceramic Jar Large" or "Ceramic Jar Small", we deem it as a match.

References

[1] Gillis, N.: The why and how of nonnegative matrix factorization. In: J. Suykens, M. Signoretto, A. Argyriou (eds.) Regularization, Optimization, Kernels, and Support Vector Machines. Chap- man & Hall/CRC, Machine Learning and Pattern Recognition Series (2014)

[2] Y.F. Hu, Y. Koren, and C. Volinsky, Collaborative Filtering for Implicit Feedback Datasets, Proc. IEEE Intl Conf. Data Mining (ICDM 08), IEEE CS Press, 2008, pp. 263-272.

[3] Johnson, C. C. (2014). Logistic matrix factorization for implicit feedback data. In Advances in Neural Information Processing Systems 27: Distributed Machine Learning and Matrix Computations Workshop.