
INBOX AMERICA | DOCUMENTATION

Author: Jenny Zhang (DS Intern - Rutgers University ‘20)
Last update: May 5. 2020

A) MATCHING APP (ia-matching)

1.1 Upload File

- Users choose a csv file and submit the “uploadForm”.
- Save the original file to the static folder and Postgres database.
- Implement a more efficient method to write data into Postgres database.
    - *Change method “to_sql”: too slow even for tiny dataset*
      → to be replaced by : XXX

1.2 Match

- Get master columns from the master table.
- Read original client file using “/t” as separator, and then split columns using “|” as a separator.
- Provide possible master columns for each column in the client table.
- Make sure that the matching relationship is one to one.
- Save matching relationship to table TransCod in database.
- Save matched file to database.

1.3 History Interface

- Provide an interface to check all data files stored in the database.
- Allow users to select an existing dataset to do matching.
- Allow user to consult and delete past matched files

B) SEGMENTATION APP (ia-segmentation)

2.1 File Selection

- Select existing matched file in the database.
- Make sure that the matched file must contain the “cont_id” feature since we will do clustering on customers.

2.2 “Automatic” Segmentation

- The relevant features cannot not be automatically detected (lack of CPU).
- Therefore, we currently have to “pre-select” these features to do the automatic **K-Means** clustering.

2.3 Commercial Segmentation

- The app selects the optimum number of segments (based on the following features:
- `# of transaction` (for a particular customer)
- `Sum of items purchased` (=sum of transactions of a customer)
- Draw a bar plot using Echart.js in the resulting page.
- Display centroids information in the resulting page.

2.4 Custom Segmentation

1. Preprocess data

- Filter out records whose format is incorrect.

101501592	101501833	12/5/2015	4/22/2016	11/30/2015		CORAL GABLES	790138658	SOFAS	LOUNGE C	LES CONTI	SCENARIO	LOUNGE C	LEATHER
101200986	101201190	NULL	10/10/2018	10/2/2018	33160	AVENTURA	101200169	ADDTIONAL FU	NULL	NULL	BUNKY BO	BUNKY BOARD KING S	
101500305	101500963	3/22/2014	3/22/2014	3/22/2014	33154	CORAL GABLES	790027869	MEUBLE MEUB	MEUBLE	LES CONTI	QUADRO (	CONSOLE	WOOD
100004587	100005034	11/26/2011	12/21/2011	11/26/2011	10018	MADISON	790085977	SOFAS	SECTIONA	LES CONTI	SCRIPT	3-SEAT 1-/	LEATHER
100010247	100011080	7/31/2015	3/3/2016	7/18/2015	10003	MADISON	100057303	DECO OBJETS I	NULL	LES CONTI	CUSTOM F	TEA ROOM	
102500048	102501148	10/18/2019	NULL	10/18/2019	10128	UPPER WEST SID	790150758	BEDS	HEADBOA	LES CONTI	COURCHE	HEADBOA	LEATHER
100101245	100102251	6/12/2017	7/20/2017	6/11/2017	10069	MANHASSET	790129773	ACCESSORIES	DECORATI	NOUVEAU	GRAND H(	MIRROR 1	FINISH
17/10/2018	13865.23	TOM@TOMVITALED	10011	13865.23	10002	Marco A.	LAG	BRIDGE - OILEI	FABRICS	GAND FAB	Referral	NEW YORI	22/01/201
100101924	100103429	8/22/2019	9/3/2019	8/22/2019	11576	MANHASSET	790109001	COCKTAIL TABI	FIXED CO(	LES CONTI	OVNI	ROUND C(	NATURAL
100012385	100013957	1/28/2017	8/31/2017	1/28/2017	10024	MADISON	100100299	SALON SIEGES	NULL	EXTERNE	FABRIC	FABRIC	
100013670	100018169	11/24/2018	2/14/2019	11/24/2018	10069	MADISON	790206635	SOFAS	MODULAF	LES CONTI	MAH JON(	SEAT CUSI	FABRICS
101501053	101501272	9/29/2014	12/30/2014	9/29/2014	33134	CORAL GABLES	790085069	SOFAS	OTTOMAN	NOUVEAU	MADEOS	SQUARE C	LEATHER
101106386	101101111	10/9/2014	1/22/2015	10/8/2014	20854	WASHINGTON	790138649	SOFAS	SECTIONA	LES CONTI	SCENARIO	3-SEAT 1-/	LEATHER
100015520	100017844	10/18/2018	NULL	10/18/2018		MADISON	790206638	SOFAS	MODULAF	LES CONTI	MAH JON(	SEAT CUSI	FABRICS
100001058	100000937	7/19/2007	8/7/2007	7/19/2007	10017	MADISON	900007516	SALON SIEGES	SALON DR	LES CONTI	MAH JON(	CUSHION	126

2. Feature selection & new table

1. The user will be offered a pre-selection of features which are considered by IA easier to process and suitable for clustering (see RMD headers below).

   *'cont_id'(mandatory),`prod_family`, `prod_subfamily`, `prod_style`, `store_id`, `prod_price_net`(*the sum of transaction amount*), `transaction_id`(*the sum of transactions*)*

   This is because almost all columns in the dataset are categorical features. In order to make the dataset suitable for the K-means clustering model to run on, we have to do one-hot encoding on these categorical features, which might lead to a very huge dataset since every distinct option of a categorical feature is a new column.

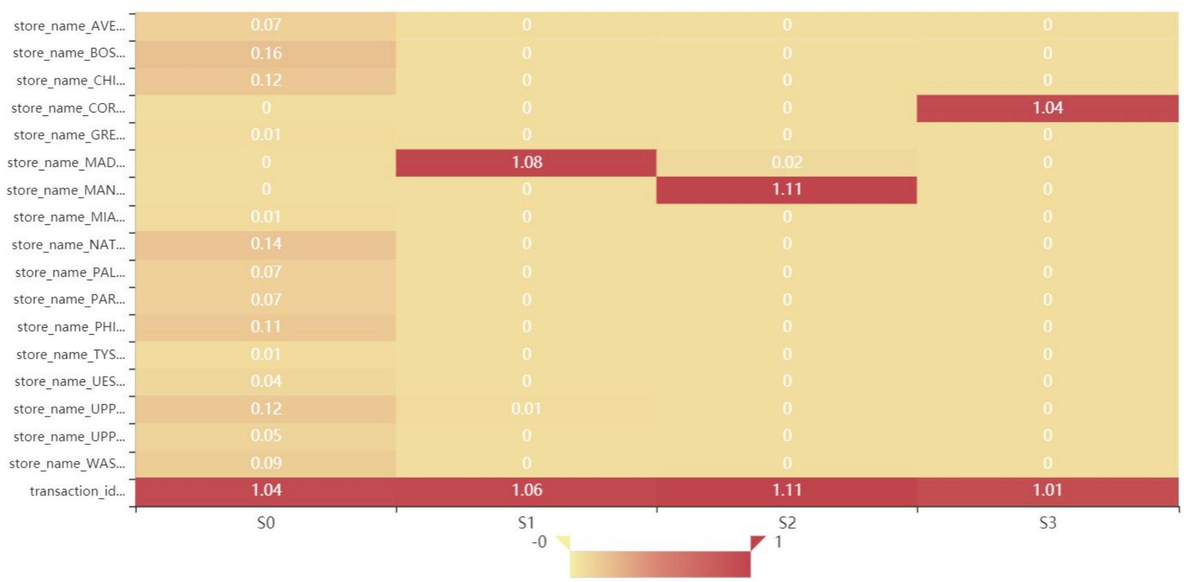
2. For example, if the user selects `*prod_family*`, we can group the records by customer id, and check all product families purchased by each customer and do binary encoding (1,0). We could also decide to calculate the sum of transactions and purchases.

3. A new table is generated based on selected features.

4. The clustering is performed, the results can be visualised with scattered plots.

Customer_id	***COMPLEMENTARY PIECES***	***OFFICES***	***ACCESSORIES***	**...**
100504335	1	0	1	…

a. Draw a heat map in the resulting page.



C) FILTER APP (ia-filter)

- Clean data
- Use QueryBuilder to add rules
- Allow users to name the filtered table
- Save the filtered table into database

D) RECOMMENDATION APP (jenny-predict01, jenny-predict-frontend, jenny-user-cf)

1. Database set-up
    - Apply filter to remove unavailable products

Products	1.	containing 'SERVICE'
	2.	bought less than 'm' times (m for minimum, it's a user-defined variable)
Records	1.	'*cont_id*' is Nan or contains non-digit characters.
	2.	id columns('line_id', 'cont_id', 'sales_id', 'transaction_id', prod_id', 'store_id') contain non-digit characters.
	3.	zip columns('cont_residency_zip','cont_company_zip') contain non-digit characters
	4.	phone columns('store_phone','cont_cellphone') contain non-digit characters.
	5.	date columns('cont_initial_contact_date', 'transaction_date', 'payment_date', 'delivery_date', 'date_of_record') don't contain '-'.
	6.	numerical columns('transaction_amount_tax', 'transaction_amount_net','prod_price_tax', 'prod_price_net') contain non-digit characters.

    - Save the original dataset and drop down menu table to database

2. Split original data set
    - Sort purchasing records by "cont_id" and "transaction_date"
    - Take the first ⅔ orders as past purchasing history(x), and take the rest ones as target variable(y)

    Item-cf
    - Generate a new table where each row represents an item and each column represents a customer.

    User-cf
    - Generate a new table where each row represents a customer and each column represents an item.

3. Popularity-model

    1) Identify for all customers, the frequency of purchases for:
    prod_family, prod_subfamily, proc_category, prod_model.

2) Then for each customer:
   a) Identify the # of past transaction(T).
   b) Use the distribution found in (1) to improve the odds of relevant recommendations.
      For instance, if the family 'SOFA' represents 44% of first purchases then if P=0 for a given customer, the model should reflect a weight of 44% for the SOFA family.

	T=1	**T=2**	**….**	**T=n**
Family1	**%**			
Family2				
Family3				
…..				
Familyn				

4. KNN model

Item-cf	The main idea is to make predictions based on similarity between items. Select K nearest items for each product bought by a certain customer and pick the most frequent ones as the recommendation result for him/her.
User-cf	The main idea is to make predictions based on similarity between users. Select K neighbours for each customer based on the purchasing history. Count products bought by nearest neighbours and pick the most frequent ones as a recommendation result.

5. Validation / Input of New Suggestions
   Item-cf
   Adds each suggestion as a new transaction.
   By default, the weight of each new suggestion is set to 1.
   User-cf
   Adds each suggestion as a new transaction.
   By default, the weight of each new suggestion is set to 1.

7. Comparison

@ = # of predictions for each room (Living, Bedroom, Dining room)	**METRICS**	**ITEM-CF**	**USER-CF**
@=1	Time efficiency (sec per customer)	0.2	0.3
	Precision	0.07864383082838168	0.23219098960338852
	Recall	0.08376768428890544	0.24033479473893982
@=3	Time efficiency	0.2	0.4
	Precision	0.21282401091405184	0.5015974440894568
	Recall	0.07757334659373447	0.0960832313341493
@=5	Time efficiency	0.2	1.0
	Precision	0.2348804500703235	0.6058914131690412
	Recall	0.08303293971410815	0.0664401661004152

8. Possible Improvements

- Fix the front-end to display all possible products (today it displays 3 products).
- Implement autocomplete (https://pypi.org/project/autocomplete/)