Market basket insights

Phase 4- *Submission document*

Project tittle: ***Market Basket Insights***

Phase 4: ***Development part 2***

Topic:

continue building the project by performing different activities like feature engineering, model training, evaluation .

Table of Contents

Modelling Strategy

Strategy : 1

Generate Training Data (using prior_orders_data)

- For every orders in prior_orders_data, take n-1 orders of every user for feature engineering.
- nth order of every user will used to label the dependent variable i.e. reordered.

example :

let, user A have 90 orders in prior_orders_data.

- build features using 89 orders.
- based on these features we will label the data with reordered(0/1) if any of the products he brought in 89 orders appeared in his 90th order.

Generate Validation Data (Using train_orders_data)

- Now that our training data is generated using prior_orders_data, we could leverage train_orders_data (which contains 1 order per user) to test our trained model
- we will predict the product reorder probability using trained model to give accuracy.
- Then we will pick top probable products whose probability of reordering was high and which can maximize F1-Score.
- We will use faron's f1 optimization code to do this.
- check the actual F1 score , and calculated F1 score. This will give us an idea on how effective the model is .

Generate Test Data (from orders.csv with eval ==’test’)

- Add features built on training data , based on orders and users
- For every order and product predict if it is reordered(0/1)
- Then we will pick top probable products whose probability of reordering was high and which can maximize F1-Score.
- We will use faron's f1 optimization code to pick products maximizing f1 score.

Feature engineering:

Association rule is the process of engineering data into a predictive feature that fits the requirements (and

/ or improves the performance) of a machine learning model.

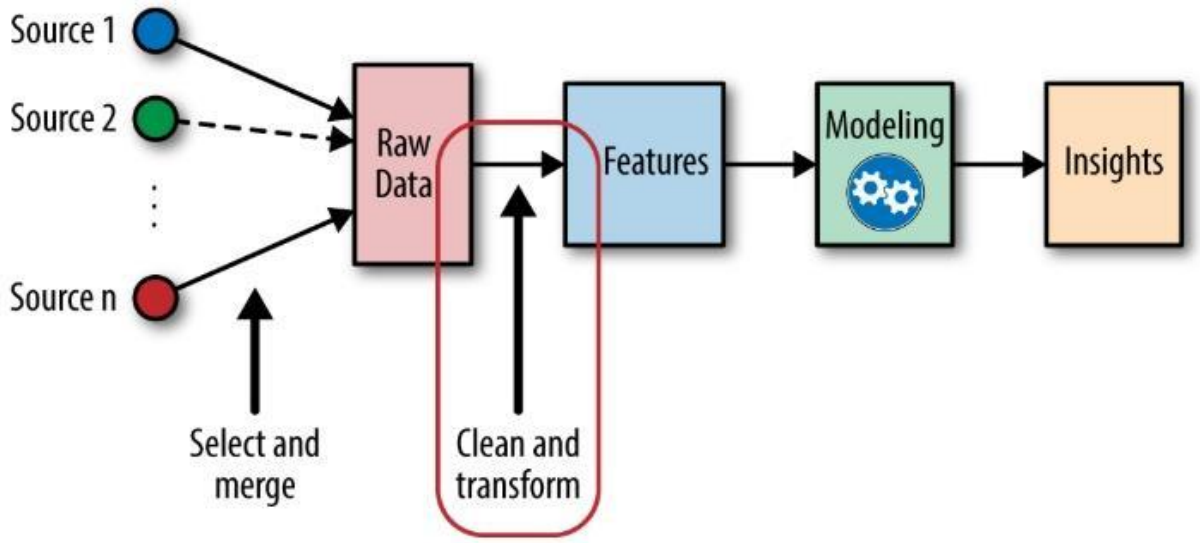


Fig: feature engineering

we want to predict if ,

User A →will buy Product B →in his next order C → reordered(1/0) ?

User ID	Product ID	Product only features	User only features	User product features	Misc features	Future Order ID	reordered ?

Dataset structure

This Future Order ID is obtained from train orders and test orders in orders.csv.

This structure is inspired by **Symeon Kokovidis 's** [**kernel**]

Generate product only features

feat_1 : product_reorder_rate : How frequently the product was reordered regardless the user preference ?

feat_2 : average_pos_incart : Average position of product in the cart ?

next 3 values are calculated based product being

- isorganic
- isYogurt — aisle
- produce — department
- isFrozen — department
- isdairy — department
- isbreakfast — department
- issnack — department
- isbeverage — department

```
##checking what's the most common word in products

from collections import Counter
x = products.product_name.values.tolist()
x = " ".join(x).split()
Counter(x).most_common()
```

```
[('&', 5203),
 ('Organic', 4903),
 ('Chocolate', 2401),
 ('Cheese', 1990),
 ('Free', 1819),
 ('Chicken', 1494),
 ('Original', 1436),
 ('with', 1432),
 ('Cream', 1241),
 ('Sauce', 1190),
 ('Yogurt', 1123),
```

out of 49688 products 4903 products are organic

These features are picked as they were most reordered product type / aisle/ dept. Now these values are then reduced to 3 columns using Non-Negative Matrix Factorization, to reduce sparsity

feat_3 : p_reduced_feat_1 : column 1 from NMF output

feat_4 : p_reduced_feat_2 : column 2 from NMF output

feat_5 : p_reduced_feat_3 : column 3 from NMF output

feat_6 : aisle_reorder_rate : How frequently a product is reordered from the aisle to which this product belongs

feat_7 : department_reorder_rate : How frequently a product is reordered from the department to which this product belongs

oduct_id	product_reorder_rate	avg_pos_incart	p_reduced_feat_1	p_reduced_feat_2	p_reduced_feat_3	aisle_id	department_id	aisle_reorder_rate	dept_reorder_r
1	0.613391	5.801836	1.0	0.0	0.0	61	19	0.548698	0.0433
2	0.133333	9.888889	0.0	0.0	0.0	104	13	0.152391	0.0319
3	0.732852	6.415162	0.0	0.0	1.0	94	7	0.527615	0.0009
4	0.446809	9.507599	0.0	1.0	0.0	38	1	0.556655	0.0060
5	0.600000	6.466667	0.0	0.0	0.0	5	13	0.280627	0.0319
...	...	...	...	...	...	...	...	...	
49684	0.111111	4.333333	0.0	1.0	0.0	124	5	0.572344	0.0399
49685	0.122449	9.571429	0.0	1.0	0.0	42	1	0.542171	0.0060
49686	0.700000	7.500000	0.0	0.0	1.0	112	3	0.670168	0.0038
49687	0.461538	7.538462	0.0	0.0	1.0	41	8	0.620883	1.7632
49688	0.168539	10.000000	0.0	0.0	0.0	73	11	0.316871	0.3276

product only features

Generate User only Features

feat_1 : user_reorder_rate : What is the average reorder rate on orders placed by a user ?

feat_2 : user_unique_products : What is the count of distinct products ordered by a user?

feat_3 : user_total_products : Count of all products ordered by a user?

feat_4 : user_avg_cart_size : Average products per order by a user ? = average cart size ?

feat_5 : user_avg_days_between_orders : Average number of days between 2 orders by a user ?

feat_6 : user_reordered_products_ratio : number of unique products reordered / number of unique products ordered

	user_id	user_reorder_rate	user_unique_products	user_total_products	user_avg_cart_size	user_avg_days_between_orders	user_reordered_products_ratio
0	1	0.694915	18	59	5.900000	17.600000	0.555556
1	2	0.476923	102	195	13.928571	14.142857	0.362745
2	3	0.625000	33	88	7.333333	11.083333	0.575758
3	4	0.055556	17	18	3.600000	11.000000	0.058824
4	5	0.378378	23	37	9.250000	10.000000	0.347826
...	...	...	...	...	...	...	...
206204	206205	0.250000	24	32	10.666667	13.333333	0.000000
206205	206206	0.473684	150	285	4.253731	3.716418	0.000000
206206	206207	0.587444	92	223	13.937500	13.437500	0.000000
206207	206208	0.707533	198	677	13.816327	7.285714	0.000000
206208	206209	0.472868	68	129	9.923077	16.153846	0.000000

User only features

Generate User Product Features

Now that we have created product only and user only features , we will now create features based on how user interacts with a product

feat_1 : u_p_order_rate : How frequently user ordered the product ?

feat_2 : u_p_reorder_rate : How frequently user reordered the product ?-

feat_3 : u_p_avg_position : What is the average position of product in the cart on orders placed by user ?

feat_4 : u_p_orders_since_last : What's the number of orders placed since the product was last ordered ?

feat_5 : max_streak : Number of orders where user

	user_id	product_id	u_p_order_rate	u_p_reorder_rate	u_p_avg_position	u_p_orders_since_last	max_streak
0	1	196	0.169492	0.900000	1.400000	0	6
1	1	10258	0.152542	0.888889	3.333333	0	5
2	1	10326	0.016949	0.000000	5.000000	5	0
3	1	12427	0.169492	0.900000	3.300000	0	6
4	1	13032	0.050847	0.666667	6.333333	0	1
...	...	...	...	...	...	...	...
13307948	206209	43961	0.023256	0.666667	8.000000	1	1
13307949	206209	44325	0.007752	0.000000	8.000000	6	0
13307950	206209	48370	0.007752	0.000000	8.000000	2	0
13307951	206209	48697	0.007752	0.000000	6.000000	6	0
13307952	206209	48742	0.015504	0.500000	9.000000	1	0

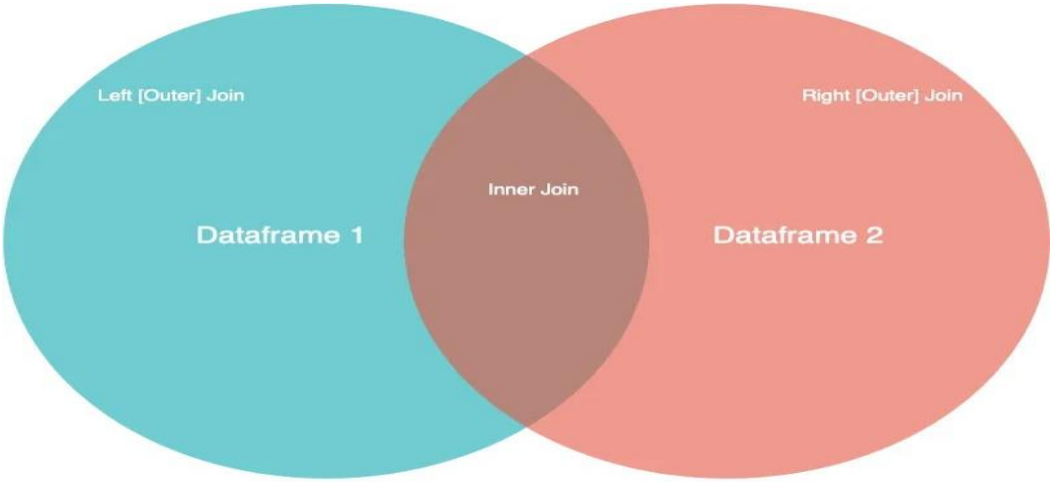
13307953 rows × 7 columns

continuously brought a product without miss

user product features

Merge above features :

Now merge these independent features (user only features , product only features, and user — product features) → call it as merged_df .



left_on=['user_id'], right_on = ['user_id'])

merged_df = pd.merge(merged_df, product_features, how='outer', left_on=['product_id'], right_on = ['product_id'])

Now , this dataframe will contain feature for all possible user — product pairs, some of which will be used in training models (using train orders) and testing (using test orders)

Misc Features

a. **Product features based on time**

def product_time(prior_data = None):

"""

feature : how frequently product was reordered on any given hour ?

```
    """
    df = prior_data.groupby(['product_id','order_hour_of_day'])["reordered"].size()
    df = df/prior_data.groupby(["product_id"]).size()
    df = df.reset_index(name = 'hour_reorder_rate')
    return df
```

	product_id	order_hour_of_day	hour_reorder_rate
0	1	0	0.006479
1	1	1	0.005940
2	1	2	0.004860
3	1	3	0.002700
4	1	4	0.002160
...	...	...	...
767863	49688	19	0.056180
767864	49688	20	0.044944
767865	49688	21	0.033708
767866	49688	22	0.011236
767867	49688	23	0.011236

767868 rows × 3 columns

order_hour_of_day features

feature : reorder frequency of a product given any hour of the day

b. Product features based on day of week

```
def product_day(prior_data = None):
```

```
    """
    feature: how frequently product was reordered on any given day ?
    """
    df = prior_data.groupby(['product_id','order_dow'])["reordered"].size()
    df = df/prior_data.groupby(["product_id"]).size()
    df = df.reset_index(name = 'day_reorder_rate')
  return df
```

	product_id	order_dow	day_reorder_rate
0	1	0	0.108531
1	1	1	0.215983
2	1	2	0.146328
3	1	3	0.141469
4	1	4	0.159287
...	...	...	...
317897	49688	2	0.134831
317898	49688	3	0.123596
317899	49688	4	0.123596
317900	49688	5	0.101124
317901	49688	6	0.112360

317902 rows × 3 columns

feature: What is the reorder frequency of any product given any day of week ?

C.Product features based on difference between 2 orders

d. User feature based on difference between 2 orders

e. User — product reorder rate based on difference between 2 orders

Generate Training Data and Test data

```
#get train orders


upd_train_orders =
train_order_data[['user_id','order_id','product_id','reord
ered']]

last_orders =
upd_train_orders.groupby(['user_id'])['order_id'].max().r
eset_index(name = 'new_order_id')

order_details =
train_order_data[['order_id','order_dow','order_hour_of
_day','days_since_prior_order']]

order_details = order_details.drop_duplicates()


#merge latest orders (prior_last_orders) with above user
and product feat
```

```
train_orders_merged_df = pd.merge(merged_df,
upd_train_orders,  how='left',
left_on=['user_id','product_id'], right_on =
['user_id','product_id'])

train_orders_merged_df =
pd.merge(train_orders_merged_df, last_orders, on =
'user_id')

train_orders_merged_df.drop("order_id", axis = 1,
inplace = True)

train_orders_merged_df.rename(columns =
{'new_order_id':'order_id'}, inplace = True)

train_orders_merged_df =
pd.merge(train_orders_merged_df, order_details, on =
'order_id')

train_orders_merged_df[['reordered']]=
train_orders_merged_df[['reordered']].fillna(value=0.0)


#merge with misc features
```

```
train_orders_merged_df =
pd.merge(train_orders_merged_df, hour_reorder_rate,
on=['product_id','order_hour_of_day'], how = 'left')

train_orders_merged_df[['hour_reorder_rate']]=
train_orders_merged_df[['hour_reorder_rate']].fillna(val
ue=0.0)


train_orders_merged_df =
pd.merge(train_orders_merged_df, day_reorder_rate,
on=['product_id','order_dow'], how = 'left')

train_orders_merged_df[['day_reorder_rate']]=
train_orders_merged_df[['day_reorder_rate']].fillna(valu
e=0.0)


train_orders_merged_df =
pd.merge(train_orders_merged_df,
p_days_since_prior_order_reorder_rate,
on=['product_id','days_since_prior_order'], how = 'left')

train_orders_merged_df[['p_days_since_prior_order_reo
rder_rate']]=
train_orders_merged_df[['p_days_since_prior_order_reo
rder_rate']].fillna(value=0.0)
```

```
train_orders_merged_df =
pd.merge(train_orders_merged_df,
u_days_since_prior_order_reorder_rate,
on=['user_id','days_since_prior_order'], how = 'left')

train_orders_merged_df[['u_days_since_prior_order_reo
rder_rate']]=
train_orders_merged_df[['u_days_since_prior_order_reo
rder_rate']].fillna(value=0.0)


train_orders_merged_df =
pd.merge(train_orders_merged_df,
days_since_prior_reorder_rate,
on=["user_id","product_id",'days_since_prior_order'],
how = 'left')

train_orders_merged_df[['days_since_prior_reorder_rate
']]=
train_orders_merged_df[['days_since_prior_reorder_rate
']].fillna(value=0.0)
```

This code will merge the merged_df (from above) with the train orders data,

similarly , we will merge test data with merged_df(from above) to generate test data.

lets have a sneak peek at our training data and test data

user_id	product_id	u_p_c	u_p	u_p_	u_p	max	user_	user	user	user	user	produ	avg_	p_red	p_rec	p_re	aisle_	depar	aisle_reor	dept_reor	order_do	order_ho	days_sinc	hour_reor	day_reorc	p_days_si	u_days_si	days_sinc	order_id	reordered
1	196	0.17	0.9	1.4	0	6	0.7	18	59	5.9	18	0.6	0.776	3.7	0	0	1	77	7	0.6387	0.000944	4	0	14	0.059652	0.15258	0.038725	0.101695	0.1	1187899	1
1	10258	0.15	0.9	3.3	0	5	0.7	18	59	5.9	18	0.6	0.714	4.3	0	0	0	117	19	0.519	0.04333	4	0	14	0.062179	0.137718	0.04111	0.101695	0.111111	1187899	1
1	10326	0.02	0	5	5	0	0.7	18	59	5.9	18	0.6	0.652	4.2	0	0	0	24	4	0.7183	0.00518	4	0	14	0.066956	0.123959	0.035831	0.101695	0	1187899	0
1	12427	0.17	0.9	3.3	0	6	0.7	18	59	5.9	18	0.6	0.741	4.8	0	0	0	23	19	0.592	0.04333	4	0	14	0.065318	0.130173	0.041847	0.101695	0.1	1187899	0
1	13032	0.05	0.7	6.3	0	1	0.7	18	59	5.9	18	0.6	0.657	5.6	0	0	0	121	14	0.572	0.318	4	0	14	0.061584	0.144495	0.032791	0.101695	0	1187899	1
1	13176	0.03	0.5	6	5	1	0.7	18	59	5.9	18	0.6	0.833	5.1	0	0	0	24	4	0.7183	0.00518	4	0	14	0.058385	0.117549	0.02847	0.101695	0	1187899	0
1	14084	0.02	0	2	9	0	0.7	18	59	5.9	18	0.6	0.811	5.8	0	0	0	91	16	0.6924	0.000732	4	0	14	0.055538	0.117917	0.032193	0.101695	0	1187899	0
1	17122	0.02	0	6	5	0	0.7	18	59	5.9	18	0.6	0.676	6.3	0	0	0	24	4	0.7183	0.00518	4	0	14	0.069885	0.120317	0.035879	0.101695	0	1187899	0
1	25133	0.14	0.9	4	0	1	0.7	18	59	5.9	18	0.6	0.74	7	0.699	0.72	0	21	16	0.5854	0.000732	4	0	14	0.060362	0.124919	0.030504	0.101695	0.125	1187899	1
1	26088	0.03	0.5	4.5	8	1	0.7	18	59	5.9	18	0.6	0.539	6.5	0.699	0.72	0	23	19	0.592	0.04333	4	0	14	0.049544	0.129608	0.034879	0.101695	0	1187899	1
1	26405	0.03	0.5	5	6	1	0.7	18	59	5.9	18	0.6	0.441	3.1	0	0	0	54	17	0.528	0.531	4	0	14	0.066722	0.147446	0.023888	0.101695	0	1187899	1

Training data

user_id	produ	u_p_	u_p	u_p_	u_p_	ma	user_	user_	user_total	user_avg_	user_avg_	user_reor	produ	avg_pc	p_re	p_re	p_re	aisle_i	depar	aisle	dep	order_	orde	days	hour_	day	p_day:	u_day:	days_s	order_id
3	248	0.01	0	3	10	0	0.625	33	88	7.332	11.086	0.5757	0.4	10.62	1	0	0	117	19	0.5	0	5	1	11	0.09	0	0.022	0.057	0	2774568
12	248	0.01	0	22	0	0	0.176	61	74	14.8	20	0.1476	0.4	10.62	1	0	0	117	19	0.5	0	1	2	30	0.04	0	0.096	0.527	1	1356845
418	248	0.01	0	3	3	0	0.359	107	167	20.88	20.62	0.3738	0.4	10.62	1	0	0	117	19	0.5	0	6	3	13	0.05	0	0.025	0.138	0	3073553
503	248	0	0	13	22	0	0.614	129	334	9.28	9.78	0.2015	0.4	10.62	1	0	0	117	19	0.5	0	1	1	8	0.08	0	0.063	0.177	1	1490499
720	248	0.01	1	4.3	2	1	0.618	118	309	13.44	13.78	0.1356	0.4	10.62	1	0	0	117	19	0.5	0	1	1	19	0.08	0	0.01	0	0	391588
1096	248	0.01	0	6	0	0	0.468	66	124	6.527	7.42	0.379	0.4	10.62	1	0	0	117	19	0.5	0	0	2	6	0.04	0	0.079	0.056	0	1295088
1109	248	0.01	1	8	4	1	0.709	100	343	9.02	8.734	0.05	0.4	10.62	1	0	0	117	19	0.5	0	5	1	4	0.08	0	0.063	0.02	0	1601439

As we can see , we do not have reordered column for test data (we will predict that).

Training Models

This will be comparison study based on different approaches, and selecting the best performing one. For each models, we decide performance based on score on kaggle and logloss.
We will use 2 approaches to get results,
a. Global Threshold (0.18 , 0.19, 0.2):
These global thresholds are selected based on:

Adhoc approach: Uploaded many submission files using different thresholds, and saw that after 0.2 F1- score started to decrease.

Using strategy 1, discussed above: we tested on train_orders, to get an global thresholds

```
Gridsearch for CV:
Threshold :0.1 - Mean F1 : 0.39482614483927203
Threshold :0.11 - Mean F1 : 0.40821274814817676
Threshold :0.12 - Mean F1 : 0.41921852708056134
Threshold :0.13 - Mean F1 : 0.42838934380977734
Threshold :0.14 - Mean F1 : 0.43585812272191155
Threshold :0.15 - Mean F1 : 0.441635249877309
Threshold :0.16 - Mean F1 : 0.44591584719058164
Threshold :0.17 - Mean F1 : 0.44899811372540854
Threshold :0.18 - Mean F1 : 0.45027007949260434
Threshold :0.19 - Mean F1 : 0.45107478024152 75
Threshold :0.2 - Mean F1 : 0.4508590730312771
Threshold :0.21 - Mean F1 : 0.4496127850119758
Threshold :0.22 - Mean F1 : 0.4479275352903583
Threshold :0.23 - Mean F1 : 0.4454477283 5112206
Threshold :0.24 - Mean F1 : 0.4420615597988679
Threshold :0.25 - Mean F1 : 0.43846782715130017
Threshold :0.26 - Mean F1 : 0.4340069375869638
Threshold :0.27 - Mean F1 : 0.4291958342288445
Threshold :0.28 - Mean F1 : 0.42408857866299154
Threshold :0.29 - Mean F1 : 0.41858467669934 42
Threshold :0.3 - Mean F1 : 0.41263509780995516
Threshold :0.31 - Mean F1 : 0.406413121136945
```

As seen above Mean F1 — drops after probability threshold 0.2. and highest were 0.18, 0.19, 0.2

For every model, we will generate 3 results (submission files) for above thresholds.

We will pick only those products whose predicted reorder probability ≥ given threshold else None.

b. Local Threshold (F1 - Maximization)

As described in last post , we will use Faron's implementation of F1-Maximization such that every order will have its own local threshold and pick those products which will maximize the F1- score.

Here are some examples , where there can be different thresholds for different orders. (examples shown here are generated to debug the model after training them)

```
X = {}
X['user_id'] = 5
recommended_products = final(X)

print()
print("="*20)
print("Recommended products")
print("="*20)
for _,value in recommended_products[0]['recommend'].items():
    print(value)
```

```
Threshold : P(X) > 0.3746
Maximum F1 : 0.5942

====================
Recommended products
====================
Organic Whole Kernel Sweet Corn No Salt Added
Semi-Soft Cheese, Ripened
Organic Red Onion
Red Raspberries
Snow Peas
Large Organic Omega3 Brown Eggs
Organic Soba
Plain Whole Milk Yogurt
Organic Blackberries
Uncured Genoa Salami
Organic Grape Tomatoes
Whole Vitamin D Milk
```

Recommendations for User ID : 5

```
X = {}
X['user_id'] = 55
recommended_products = final(X)

print()
print("="*20)
print("Recommended products")
print("="*20)
for _,value in recommended_products[0]['recommend'].items():
    print(value)
```

```
Threshold : P(X) > 0.5853
Maximum F1 : 0.7680

====================
Recommended products
====================
Imported Mineral Water
Organic Bunny Fruit Snacks Berry Patch
Chocolate Peppermint Stick Bar
Organic Sticks Low Moisture Part Skim Mozzarella String Cheese
Goldfish Cheddar Baked Snack Crackers
Organics Chocolate Milk with DHA
Original Whipped Cream Cheese
Organic Uncured Sliced Black Forest Ham
Pirate's Booty Aged White Cheddar Baked Rice and Corn Puffs
Plain Mini Bagels
Grape White/Green Seedless
```

Recommendations for User ID : 55

As seen from above examples , local threshold for every order can boost the F1 Score.

Let's train some models,

Model 1 : Logistic Regression

```
log_reg = LogisticRegression(random_state=0, n_jobs = -1)

log_reg.fit(X_train_norm,y_train)

predict_y = log_reg.predict_proba(X_val_norm)

print("logloss on validation data :",log_loss(y_val, predict_y, labels=[0,1], eps=1e-15))
```

Logloss on validation data : 0.2550918280106341

Model 2: Decision Tree

```
param_grid = {}

param_grid['max_depth'] = [5,10,15,20]

param_grid['min_samples_split'] = [2,3,4,5]


dt_clf = DecisionTreeClassifier()
```

```
r_search = RandomizedSearchCV(dt_clf,
param_distributions=param_grid, cv = 5, verbose = True,
n_jobs = -1)

r_search.fit(X_train, y_train)


predict_y = r_search.predict_proba(X_val)

print("logloss on validation data :",log_loss(y_val,
predict_y, labels=[0,1], eps=1e-15))
```

Logloss on validation data : 0.2509911734828939

Model 3: Random Forest Classifier

```
From sklearn.ensemble import RandomForestClassifier

Clf=RandomForestClassifier(n_estimators=25,random_sta
te=42,n_jobs=-1)

Clf.fit(X_train, y_train)

Sig_clf = CalibratedClassifierCV(clf, method=”sigmoid”)

Sig_clf.fit(X_train, y_train)


Predict_y = sig_clf.predict_proba(X_val)
```

Print("logloss on validation data :",log_loss(y_val, predict_y, labels=[0,1], eps=1e-15))

Logloss on validation data : 0.2518767530531320 6

Improve the model

We can improve the Catboost Model slightly

During my trials on different models , I found out that , instead of splitting our training data randomly into training and validation set for training our models, if we split training data on users , we can improve our models.

EVALUATION

A. Introduction

The previous chapter described the implementation in detail of all the sections of this research. This chapter

evaluates and justifies the results and the decision support system developed by the researcher.

B. Association Comparison

With the decision support system, we are now able to compare association rules among items or items sets with

the date or week days of the corresponding year. This comparison helps to identify consumer behavior on

particular item or items set which is our final goal. This knowledge further can be converted to marketing

information in order to conduct a promotional campaign or adjust business decisions and approaches.

Further, our approach is unique and novel; this research makes traditional market basket analysis in a

comprehensive manner.

Among over 1 million of association rules (Figure 28) which were generated from the research process makes

the complexity of the analysis is probably high. Therefore, we select random items and item sets to illustrate

knowledge discovery by comparing them.

As an example, in 2017 January, a promotion was conducted named $5 Bento Promo and the maximum support

is 1% and the confidence is limited to 25%. In 2018 January, there was a similar promotion conducted and that

shows the highest support throughout the association rules. The promotion was $5.99 Bento Promo and with

Miso Soup gives the highest support value of 0.34. This emphasizes that buyers who bought Miso Soup are having 34% probability of buying $5.99 Bento Promo. Further, it shows that in some days the confidence level reaches a maximum of 100%.

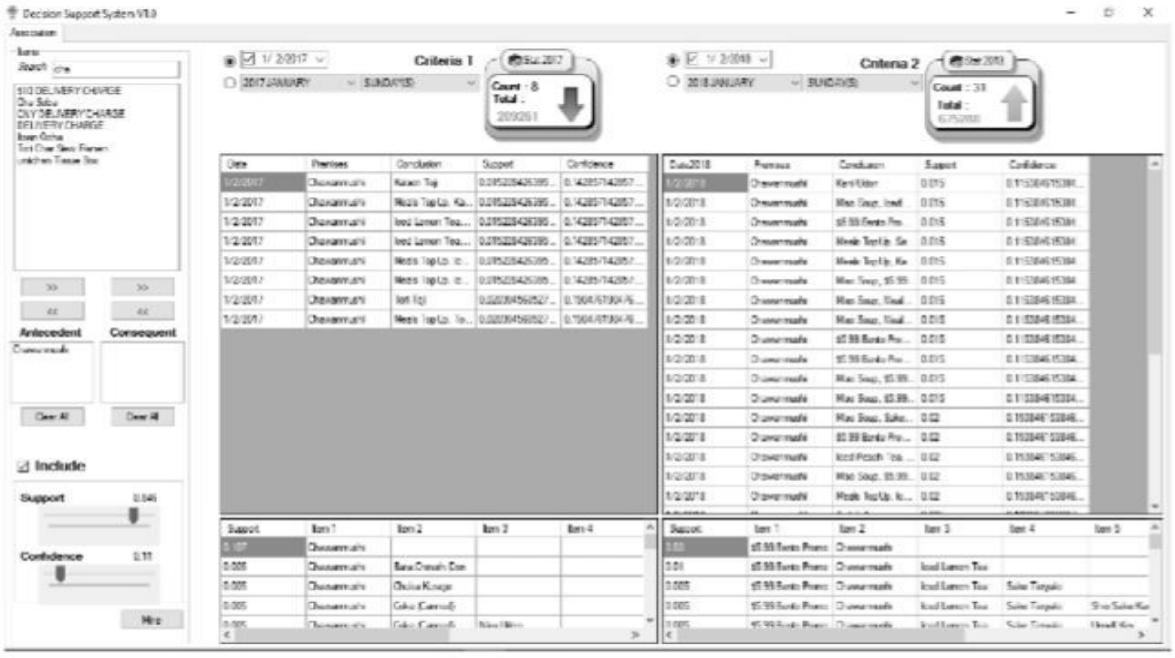
This is an advantage for the marketing team in order to predict and conduct such promotions in upcoming years.

Another example was buying Chawanmushi on Sundays in January 2017, shows support of 3% along with Tori Toji and Iced Lemon Tea. Further among all transactions, when buying Chawanmushi there is a confidence of 26.9% of buying Tori Toji and Iced Lemon Tea.

If we compare this with Sundays of January 2018, this association is spread through 2 Sundays with the support of 2%, confidence of 10% and support of 1%, confidence 13.6% respectively.

Comparison of knowledge

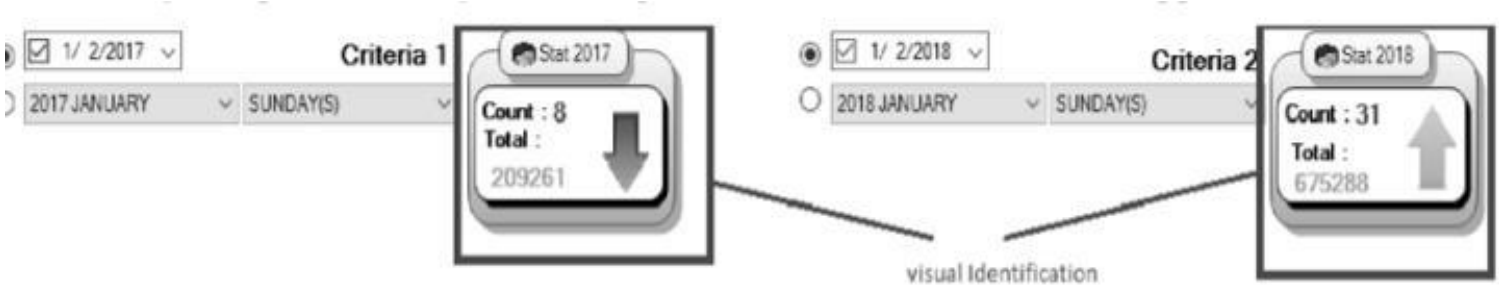
C. Evaluation of Decision support system



Proposed Decision support system can generate summarized results for association rules and the marketing

people can check for least associations and frequent item sets and conduct marketing campaigns. Therefore, we

have provided a very compact summary of the output of this research in terms of support and confidence.



As an enhanced feature, the user can easily identify the year of having the most counts. This feature allows the User to visually identify the year which having most records without manually counting the number of records. This feature is especially helping when the record count exceeds the human counting limitations and not clearly Visualized in the grid view.

It is very difficult to evaluate a system in terms of errors; bugs and we have managed to extract only the relevant

And useful information and knowledge from the executed process. This knowledge is the key to the aim and Objective of this research. Marketing people are now able to analyze these associations and then conduct Marketing campaigns and strategies for less selling item sets with low support and confidence.

Conclusion :

Market basket analysis helps the retailers know about the products frequently bought together so as to keep those items always available in their inventory. The source from which these patterns are found is the vast amount of data that is continually collected and stored.

Feature engineering, like so many things in data science, is an iterative process. Investigating, experimenting, and doubling back to make adjustments are crucial. The insights you stand to gain into the structure of your data and the potential improvements to model performance are usually well worth the effort. Plus, if you're relatively new to all this, feature engineering is a great way to practice working with and manipulating DataFrames! So stay tuned for future posts covering specific examples (with code) of how to do just that.

Algorithms used in market basket analysis. Market basket analysis utilizes association rule {IF} - > {THEN} to predict the probability of certain products being purchased together. They count the item frequency occurring together and seek to find associations that occur more than expected.

Thank you