FINAL REPORT

Model Analysis for E-Commerce Recommender Systems

Li Yifan - Stefano Zavagli - Arthur De Bono - Moyi Yang - Wu Fei

Project Motivation

Our project is attempting to address Machine Learning methods in the space of online retail sales. Customer product choice prediction is naturally one of the most popular fields nowadays, as incremental efficiencies can yield substantial profit gains for enterprises. That said, there are a plethora of variables within the business value chain where Machine Learning can bring about improvements. We will address here some of literature that has been published, and how research has affected the evolution of online retail business models.

By building different models, we will have access to the key factors permitting us to predict future purchases of customers and let them see the SKUs that they are more likely to purchase in the future. We can only recommend a few things to customers, so it is important to choose which to recommend to a specific customer. This project represented a unique opportunity of exposure to real business datasets in the e-Commerce space. The e-Commerce space is one of the fastest growing categories in retail; particularly relevant in times like these, during Covid-19. We tried a wide variety of models on this data, and chose to present the 3 most interpretable, and best performing models

Among the main motivations for our team, we wanted to pursue the project because:

- *This project represented a unique opportunity of exposure to real business datasets in the e-Commerce space*
- *The e-Commerce space is one of the fastest growing categories in retail; particularly relevant in times like the one we find ourselves in.*
- *We tried a wide variety of models on this data, and chose to present the 3 most interpretable, and best performing models, ultimately gaining a grasp and sense of which models work better with which data.*

Data Description

We used a dataset from JD.com, China's largest retailer. The dataset contains transactions associated with over 2.5 million customers (anonymized) and 31,868 SKUs (anonymized) over the month of March, 2018. A detailed view on the activities associated with all SKUs within one anonymized consumable category is provided by JD.com. Owing to confidentiality, the specific category is not disclosed. The data set consists of seven separate subsets, or tables, that are labeled as (1) skus, (2) users, (3) clicks, (4) orders, (5) delivery, (6) inventory, and (7) network.

From these seven tables, our plan is to research the relationship between customer information and the product the user will purchase in future. That is, using customer identity information and his history click and order data to predict future purchases.

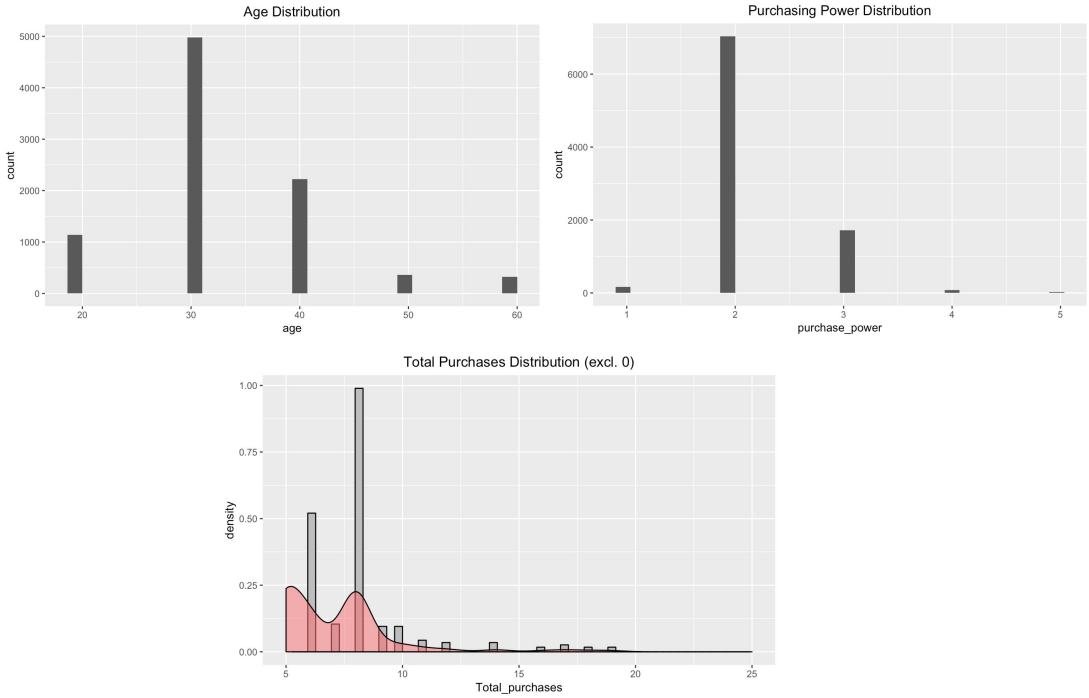


Figure 1: Data Histograms

Data Cleaning and Preprocessing

From 7 original datasets we were given, we generated two smaller and more tractable sets, which were accomplished by filtering out users who purchased fewer than 5 times and SKUs with less than 1000 orders. We split the train and test sets by the time of the order, and used the first-20 days worth of data for the training of the models, and the last 10 days of data for the testing section. The resulting dataframes that were fed into our models can be seen below:

user_ID <fctr>	user_level <int>	plus <int>	gender <int>	age <int>	marital_status <int>	education <int>	city_level <int>	purchase_power <int>
004d88db13	4	1	1	30	1	3	1	2
006bc1d136	4	1	-1	30	1	4	1	2
00e0fcf701	4	1	-1	30	1	4	2	2
00fbda605c	4	0	-1	30	0	3	1	2
016c1ee449	4	0	-1	30	1	4	2	3
02033e9b30	4	1	1	30	1	3	1	2
020c99586c	2	1	-1	30	0	3	2	2
037933dc5a	4	0	-1	30	0	3	2	2
045a30c3bb	4	1	1	30	1	4	3	2
051caaee60	3	0	1	30	0	3	1	2

1–10 of 9,020 rows Previous 1 2 3 4 5 6 ... 100 Next

Figure 2: for display purposes, we first show in this image only the 8 demographics corresponding to each user.

user_ID <fctr>	X1 <int>	X2 <int>	X3 <int>	X4 <int>	X5 <int>	X6 <int>	X7 <int>	X8 <int>	X9 <int>
004d88db13	0	0	0	0	0	0	0	0	0
006bc1d136	0	1	0	0	0	0	0	0	0
00e0fcf701	0	1	0	0	0	0	0	0	0
00fbda605c	0	1	0	0	0	0	0	0	0
016c1ee449	0	0	0	0	0	0	0	0	0
02033e9b30	0	0	0	0	0	0	0	0	0
020c99586c	0	0	0	0	1	0	0	0	0
037933dc5a	0	1	0	0	0	0	0	0	0
045a30c3bb	0	0	0	0	0	0	0	0	0
051caaee60	0	0	0	0	0	0	0	0	0

1–10 of 9,020 rows | 1–10 of 61 columns Previous 1 2 3 4 5 6 ... 100 Next

Figure 3: the SKUs are represented by: {X1, X2, X3, X4, X5, X6, X7, X8, X9}

Missing Values

There are two types of missing values. The first is the missing value under common definition. One of the first steps in our data cleaning process was to remove the observations with missing values or N/A in the features.

Calculating the average order level of a given item is an important part in data preprocessing. According to the lead time between users' first click of a given item, and their purchase, we found that oftentimes a user would determine whether to buy the item or not within the same day. Thus, using the average revenue of the item within a day can better reflect its price and discount features better. However, there are some items which are not sold in the same day that they are clicked. Such an effect can lead to missing values: for items that contain the second type of missing value, the method we used to mitigate is imputation by means of averaging the mean revenues of the day before and the day after.

Feature Engineering

The correlation matrix and vif coefficients are reported in the figure below. We can see some important characteristics with regards to pricing, as discounting and bundles seem to have influence on each other. The demographics features indicate less risks of multicollinearity at this point

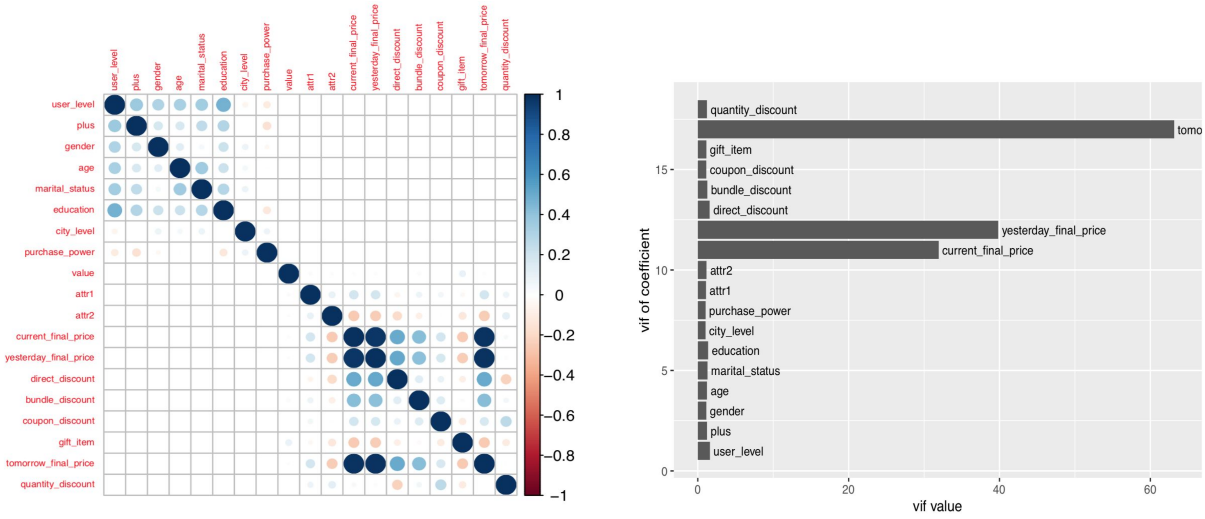


Figure 4: Mitigating multicollinearity and improving our models' performance

We tested the influence of personality towards customer preference. Using direct discounts and bundle discounts as an example, the plots in Figure 5 show the difference between plus members (the orange one is the plus member and the blue one is the non-plus number).

We know that the plus members have more chances to get a higher discount, and this fact can be seen in the direct discount graph. However, on the rightmost part of the graph of bundle discount, we can see that the non-plus members get more bundle discounts, and it means the plus members prefer a direct discount rather than a bundle discount. So we include this feature in our model.

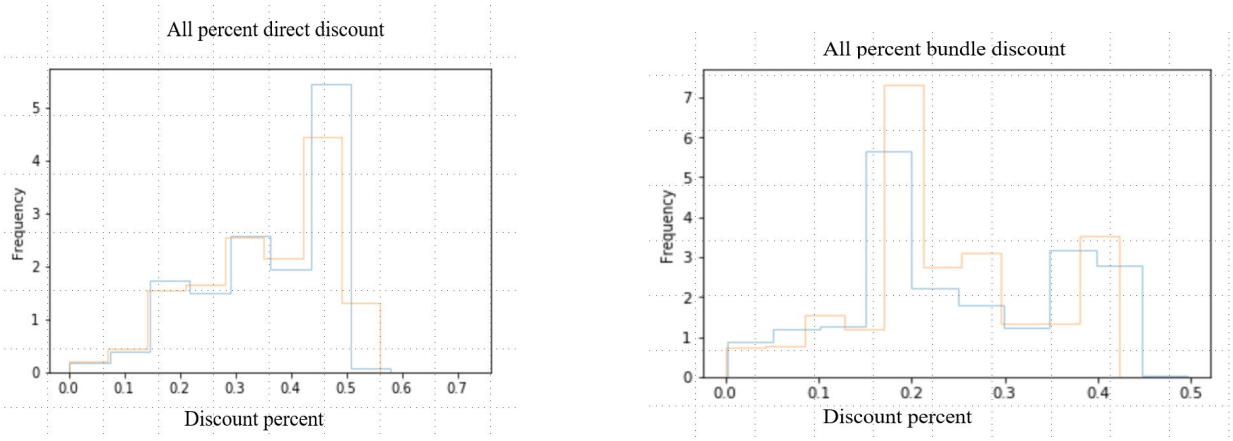


Figure 5: modeling behaviour through economic utility

ANALYTICS

Linear and Logistic Regression

One of the most basic types of ML models that we can try is a Multiple Linear Regression model. By implementing this model in R, we attempt to give weighting to specific features and create a numerical result that approaches the response variable. The MLR works better with numerical data, hence categoricals had to be discretized either into factors, or converted to continuous data via some type of rule. In terms of predicting user choice, given that we are dealing with a very sparse data matrix, as can be seen in our table, the difficulty in predicting user product choice uniquely via numerical regression proved too challenging. The best result is presented in the image below.

```
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     2.397e+00  3.698e-02  64.827  < 2e-16 ***
date            1.929e-17  8.492e-04   0.000 1.000000
user_level     -2.146e-02  6.830e-03  -3.142 0.001684 **
plus           -1.216e-01  1.080e-02 -11.261  < 2e-16 ***
gender          1.545e-03  6.095e-03   0.254 0.799887
age             2.370e-03  6.200e-04   3.823 0.000133 ***
marital_status  4.517e-02  1.173e-02   3.853 0.000118 ***
education      -7.427e-02  1.012e-02  -7.337 2.37e-13 ***
city_level      2.909e-02  4.652e-03   6.255 4.17e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4651 on 9011 degrees of freedom
Multiple R-squared:  0.03938,   Adjusted R-squared:  0.03853
F-statistic: 46.17 on 8 and 9011 DF,  p-value: < 2.2e-16
```

Figure 6: Linear Regression summary

Out-of-sample accuracy levels never surpassed the 4% mark, no matter which subsets of data we utilized, how we tuned the model, or the level of additional feature engineering that we incorporated. The LR model served simply as a reference but given the lack of potential, we continued to pursue more prolific models.

The logistic regression on the other hand, uses the data and tries to predict the probability that certain users will purchase a given item. However as the original data set has a feature that the majority of the data is not bought (more than 99%). Hence computing such probability proved extremely difficult. The end result of the logistic regression was even poorer than what we found with the Linear Regression.

Clustering and CART

Next we attempted some clustering modes. In order to find the different product groups, two types of clustering were implemented. The first type was based on grouping products that were purchased together the highest number of times. To do this, the clustering algorithm calculates the number of times a given product has been purchased together with another one. It then generates an ordered list of items. The ordered list is then processed and outputs a list of recommendations. We can say straight-away that the performance on this algorithm was not ideal, and its run-time was not very efficient either. Hence we tried the second type of clustering, the “kmeans”.

The second clustering method uses the *kmeans()* function as covered in class. After plotting the explained variation as a function of the number of clusters, the “elbow” of the curve gives a number of clusters of 3. Unfortunately, such a number of clusters is not precise enough to give an accurate recommendation for a product. Hence, we needed to use an algorithm that would be based on the item and also on customer characteristics. The two types of clustering techniques we attempted here did not achieve the goal, that is why we continued pursuing other models.

For the CART part, we constructed models that were fundamentally designed to predict each item’s purchase. Actually, for each item, the number of users with no purchases after a click is much higher than the number of users with an actual purchase. This goes back again to the sparsity of our data, and presents significant challenges for the CART model as well. Thus our prediction in almost all instances is that a user will not buy an item, and predicts so with high accuracy, naturally. The key point here was to define an appropriate loss matrix. Therefore, we needed to increase the penalty of false negatives and false positives. Once we finished the weighting exercise, we obtained total accuracy is 46.6% and the weighted TPR of 11.7%, which are substantially higher than the previous models.

Collaborative Filtering

The CF model we chose is an item-based collaborative filtering model. In the item-based CF, our main concern is of course the items and the relationships between them. The key assumption here is that if the items are similar, they are more likely to be purchased by a given user. First of all, we need to construct a function to calculate the similarities between the items. The Cosine Similarity function was used for this:

$$\text{Cosine}(X,Y) = \frac{\sum_{i=1}^{n} X_i Y_i}{\sqrt{\sum_{i=1}^{n} (X_i)^2}\sqrt{\sum_{i=1}^{n} (Y_i)^2}}$$

Then we used the item similarity matrix to make recommendations, where the top 10 neighbors similarities and the consumption records of the customer are used to calculate the relative score between the user and the item:

$$\mathrm{Score}(h,s) = \frac{\sum_{i=1}^{n} \mathrm{h}_i \times \mathrm{s}_i}{\sum_{i=1}^{n}(s_i)}$$

This score represents how strongly the user is interested in a specific item. Here is an sample of the output we have:

	X1	X2	X3	X5	X7
004d88db13	"0"	"0.0374447007661357"	"0"	"0.0697620013352452"	"0"
006bc1d136	"0"	""	"0"	"2.78372669390051"	"0"
00e0fcf701	"0"	""	"0"	"2.78372669390051"	"0"
00fbda605c	"0"	""	"0"	"2.78372669390051"	"0"
016c1ee449	"0"	""	"0"	"1.58737120020929"	"0"
02033e9b30	"0"	""	"0"	""	"0"

Figure 7: Sample of scores

Then, we just need to sort these scores and replace them with the name of its respective products. The result obtained is a matrix giving the ID user with the products (V1, V2, …, V60) recommended.

user_ID <fctr>	V1 <fctr>	V2 <fctr>	V3 <fctr>
004d88db13	X5	X60	X2
006bc1d136	X5	X48	X58
00e0fcf701	X5	X48	X58
00fbda605c	X5	X48	X58
016c1ee449	X5	X48	X58
02033e9b30	X48	X58	X29

Figure 8: Top Item Recommendations

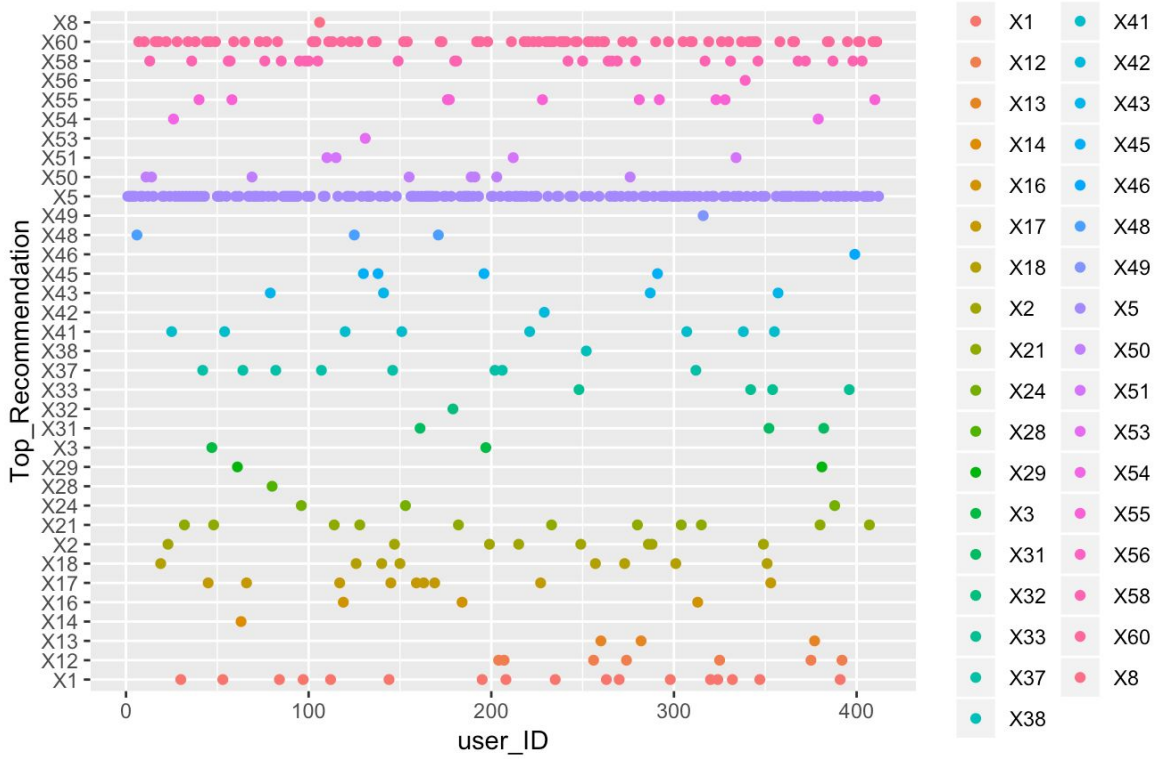


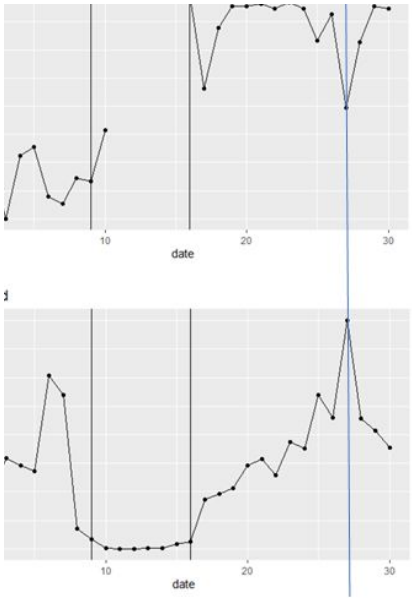
Figure 9: Top SKU recommendations for each user

Utility Based Model

This model is derived from the utility function. We processed the data and formatted the user information and sku information into two vectors: Vcu and Vsk. We have estimated the preference of customer choosing SKU1 over SKU2 by using the sigmoid function:

$$P = \frac{1}{1+e^{-x}} \qquad x = V_{cu_i}\left(V_{sk_{j_1}} - V_{sk_{j_2}}\right)$$

After visualizing the data, we remarked that the price influences the demand of customers a lot, as you can see in the picture on the right side. So we have calculated the average price of a sku in each day and included the next and previous day's price in case that there is a large promotion for a sku. As an additional input feature of the model.



Moreover, the vector we used sets some parameters in each vector in advance and, as the promotions and price of sku and each customer's personal information in advance, and learns other parameters a1 and a2 with gradient ascent, as you can see in the following pictures.

Vcu
gender
age
``
b_1
b_2

Vsk
a_1
a_2
``
percent bundle discount
percent direct discount

And with the AMAN assumption that all customers will buy the item they prefer with higher probability, we can use maximum likelihood estimation to calculate the function below.

$$L = \sum_{j_1 \in J} \sum_{j_2 \in J} \sum_{i \in I} P_{ij_1}(!P_{ij_2}) \times \ln\sigma(X_{ij_1} - X_{ij_2})$$

$$= \sum_{j_1 \in J} \sum_{j_2 \in J} \sum_{i \in I} P_{ij_1}(!P_{ij_2}) \times \ln\sigma\{Vcu_i \cdot (Vsk_{j1} - Vsk_{j2})\}$$

P_{ij}: $purchase\ \ matrix$, the amount of Sku j customer i purchase

$$!P_{ij} = \begin{cases} 0 & P_{ij} \neq 0 \\ 1 & P_{ij} = 0 \end{cases}$$

L: objective function

I: set of customer

J: set of Sku

Vcu_i: vector for customer i

Vsk_j: vector for SKU j

$\sigma(x)$: $sigmoid\ \ function$, $\frac{1}{1-e^{-x}}$

And we can use gradient ascent to optimize the result and get to the optimal value:

$$\frac{\mathrm{dL}}{\mathrm{d}Vcu_i} = \sum_{j_1 \in J} \sum_{j_2 \in J} \frac{P_{ij_1}(!P_{ij_2}) \times (Vsk_{j1} - Vsk_{j2}) \exp\{Vcu_i(Vsk_{j2} - Vsk_{j1})\}}{(1 + \exp\{Vcu_i(Vsk_{j2} - Vsk_{j1})\})}$$

$$Y_{ij_2j_1} = \frac{1}{1 + e^{Vsk_{j2} - Vsk_{j1}}}$$

P_i: the row i in P matrix

t(x): transfer of a matrix x

$$= \sum_{j_1 \in J} \sum_{j_2 \in J} P_{ij_1}(!P_{ij_2}) \times (Vsku_{j_1 j_2}) \exp\{Vcu_i(Vsku_{j_2 j_1})\} Y_{ij_2j_1}$$

$$= sum\{t(P_i)(!P_i) \times (Vsku) \exp\{Vcu_i t(Vsku)\} t(Y_i)\}$$

$$\frac{\mathrm{dL}}{\mathrm{d}Vsk_j}$$

$$= \sum_{k \in J} \sum_{i \in I} \frac{P_{ik}(!P_{ij}) \times (-Vcu_i) \exp\{Vcu_i(Vsk_j - Vsk_k)\} + (!P_{ik}) P_{ij} \times (Vcu_i) \exp\{Vcu_i(Vsk_k - Vsk_j)\}}{(1 + \exp\{Vcu_i(Vsk_{j2} - Vsk_{j1})\})}$$

$$= \sum_{k \in J} \sum_{i \in I} \frac{P_{ik}(!P_{ij}) \times (-Vcu_i) \exp\{Vcu_i(Vsku_{jk})\} + (!P_{ik}) P_{ij} \times (Vcu_i) \exp\{Vcu_i(Vsku_{kj})\}}{Y_{ij_2j_1}}$$

Finally, we can plot the accuracy of prediction depending on the number of recommendations (The black dots are the random picked recommendations).

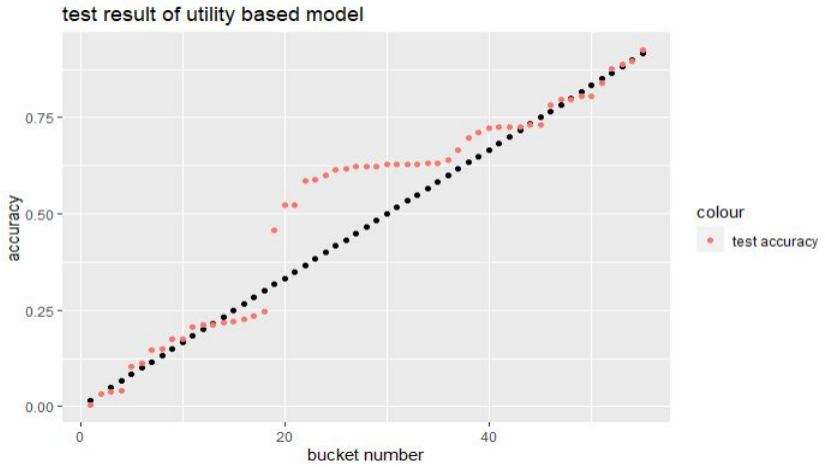


Figure 11: evaluation of model on test data

where the model performs the best is when the bucket number is between 20-40, and during this part, the recommendation accuracy is higher than baseline by about 15% on average and 20% at maximum.

IMPACT

Financial Analysis

Since the e-commerce companies like JD.com and Amazon rely on the buyers' purchase and make profit from the platform fee charged to sellers. The most significant metric among all the financial indicators is the customers' total purchase and profit rate. Compared with an e-commerce company without a recommendation system, our model can actually increase the amount of revenue by 20% from the improvement of accuracy.(From the average compared improvement of accuracy of CF model and UB model) The actual user stickiness will also increase due to the improved accuracy on recommendation and ads, therefore also enlarge the total revenue in another way.

Recently the e-commerce industry starts to focus on lower purchase level users more and more. The reason is obvious - the highland of the high purchase level customers is already a red ocean. Most of the recent uprising e-commerce companies rely on the new market of these previously ignored users. These users actually are more sensitive on the price and promotions, therefore the recommendation system is more important for them when we can promote specific items based on the user's preference.

Team Story

Our intent with the project is to explore how the methods we have covered in class fit into the JD.com's recommendation system, and examine their capability of providing useful insights. At first sight, we think that the generalized linear model, random forest and boosting will permit us to attain our goal. The objective is not to design a recommendation system outright from scratch, but rather to apply traditional machine learning models in order to understand how the product assortment problem can be improved, and assess strengths and weaknesses of different approaches under different scenarios.

Our team is focusing on the improvement of user experience and commercial efficiency in the uprising e-commerce companies. The recommendation system provided by us is a great way to boost your online shopping business by increasing the customers' purchase amount and stickiness. We only require a few categories of data and it will not violate any of the privacy regulations in your or your users' country because we only need tokenized user information. Our team consists of experienced data scientists who have worked in the industry for years and talented algorithm engineers with insight of edge technologies.

Bibliography

[1] Two Decades of Recommender Systems at Amazon.com, G. Linden ; B. Smith ; (2017)

[2] Amazon.com Recommendations, G. Linden et al (2003)

[3] R for Data Science, Garrett Grolemund and Hadley Wickham (2017)

[4] Recommendation system with cluster-based filtering of recommendations , Chan et al. 2012

[5] A mobile recommendation system based on Logistic Regression and Gradient Boosting Decision Trees, Wang et al. (2016)

[6] Annual Reports of JD: https://ir.jd.com/annual-reports

[7] Collaborative Filtering with R, Salem Marafi, (2014) (http://www.salemmarafi.com/code/collaborative-filtering-r/)

[8] Market Basket Analysis with R, Salem Marafi (2014) (http://www.salemmarafi.com/code/market-basket-analysis-with-r/)

Appendix - Collaborative Filtering and Ensemble Model

27/04/2020

```
library(pracma)
library("ggmap")
```

```
## Loading required package: ggplot2
```

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
## Google's Terms of Service: https://cloud.google.com/maps-platform/terms/.
```

```
## Please cite ggmap if you use it! See citation("ggmap") for details.
```

```
library(ggplot2)
library(jsonlite)
library(stringr)
library(tm)
```

```
## Loading required package: NLP
```

```
##
## Attaching package: 'NLP'
```

```
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
library(cluster)
library(SnowballC)
library(factoextra)
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
library(fpc)
library(maptools)
```

```
## Loading required package: sp
```

```
## Checking rgeos availability: FALSE
##      Note: when rgeos is not available, polygon geometry     computations in maptools depend on gpcl
##      which has a restricted licence. It is disabled by default;
##      to enable gpclib, type gpclibPermit()
```

```
library(maps)
```

```
##
## Attaching package: 'maps'
```

```
## The following object is masked from 'package:cluster':
##
##     votes.repub
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(caTools)
```

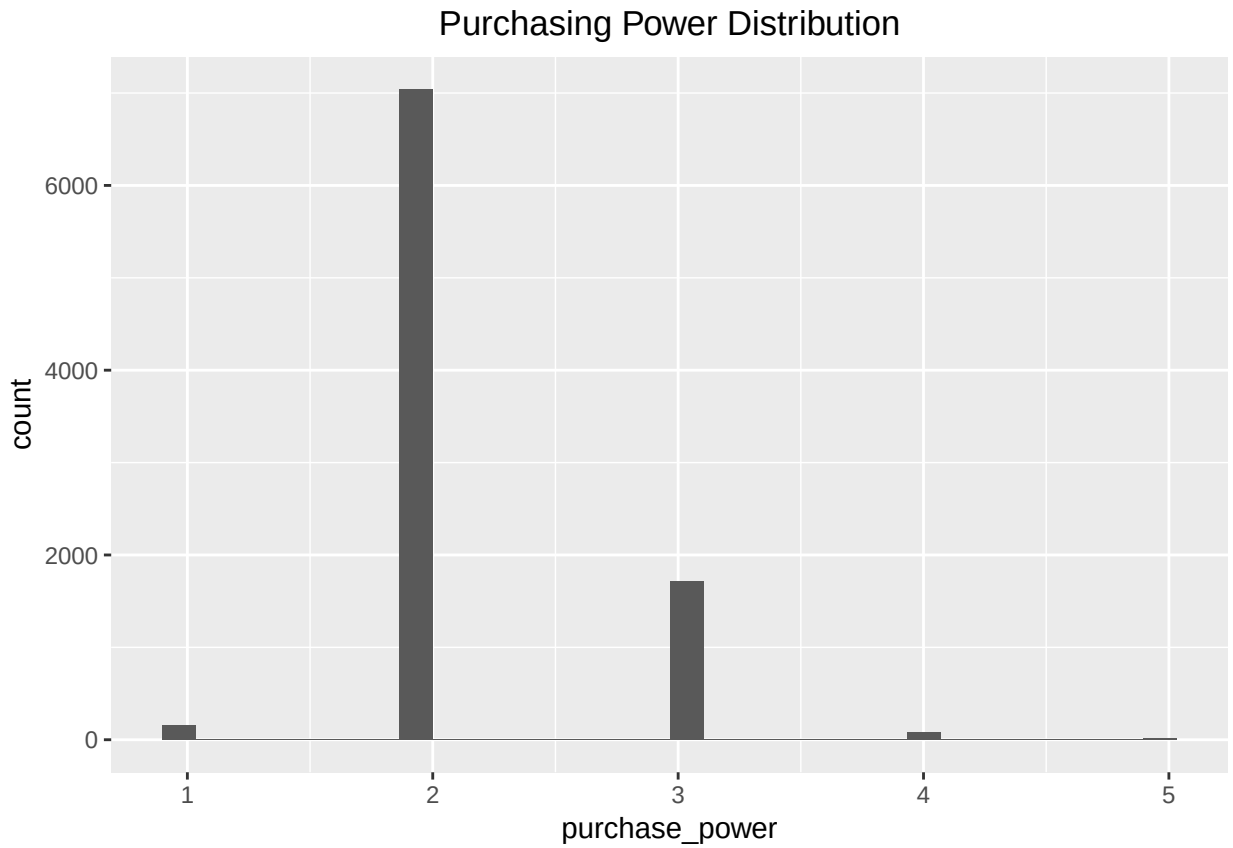
```
##
## Attaching package: 'caTools'

## The following objects are masked from 'package:pracma':
##
##     combs, trapz
```

```
setwd("/Users/stefanozavagli/Documents/Carrera/Schools/Berkeley/Clases/SPRING '20/IEOR 242/Project/Data/
train_vcuinformation <- read.csv(file = 'train_vcuinformation.csv')
test_vcuinformation <- read.csv(file = 'test_vcuinformation.csv')
```

PLOTTINGS

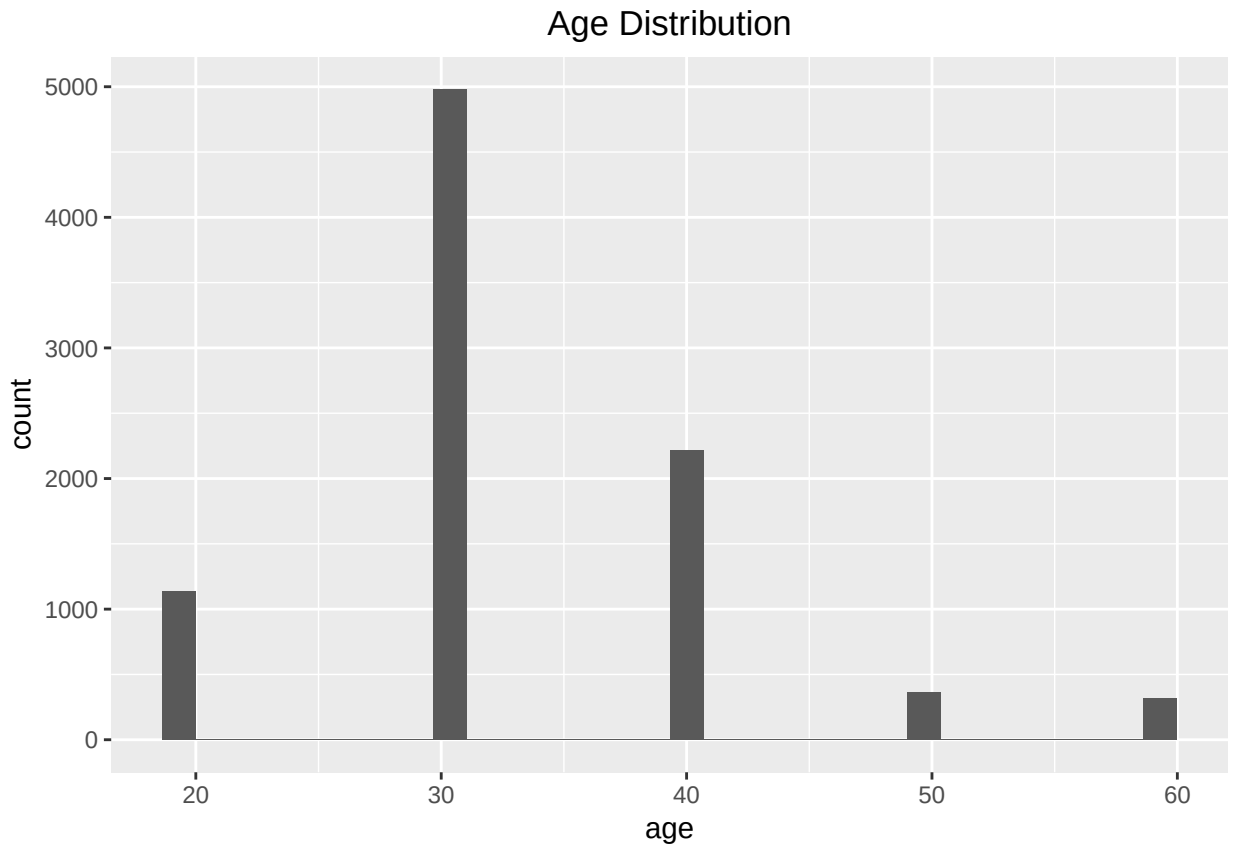
```
# train_vcuinformation[2:10]
# train_vcuinformation[,-c(1,3,4,5,6,7,8,9,10)]
# train_vcuinformation[11:20]
# plot(train_vcuinformation[11:20])
# boxplot(train_vcuinformation$age)
pphist <- ggplot(train_vcuinformation, aes(purchase_power)) +
  geom_histogram() +
  # xlab("age") +
  # ylab("frequency") +
  ggtitle("Purchasing Power Distribution") +
  theme(plot.title = element_text(hjust=0.5))

pphist
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
ahist <- ggplot(train_vcuinformation, aes(age)) +
  geom_histogram() +
  ggtitle("Age Distribution") +
  theme(plot.title = element_text(hjust=0.5))
ahist
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
users_tot_purchase = train_vcuinformation[2:3]
users_tot_purchase$user_ID = as.numeric(seq(1, length(users_tot_purchase$user_ID)))
names(users_tot_purchase)[2] = "Total_purchases"
for (i in 1:dim(users_tot_purchase)[1]) {
  users_tot_purchase[i, 2] = sum(train_vcuinformation[i, 11:70])
}
```

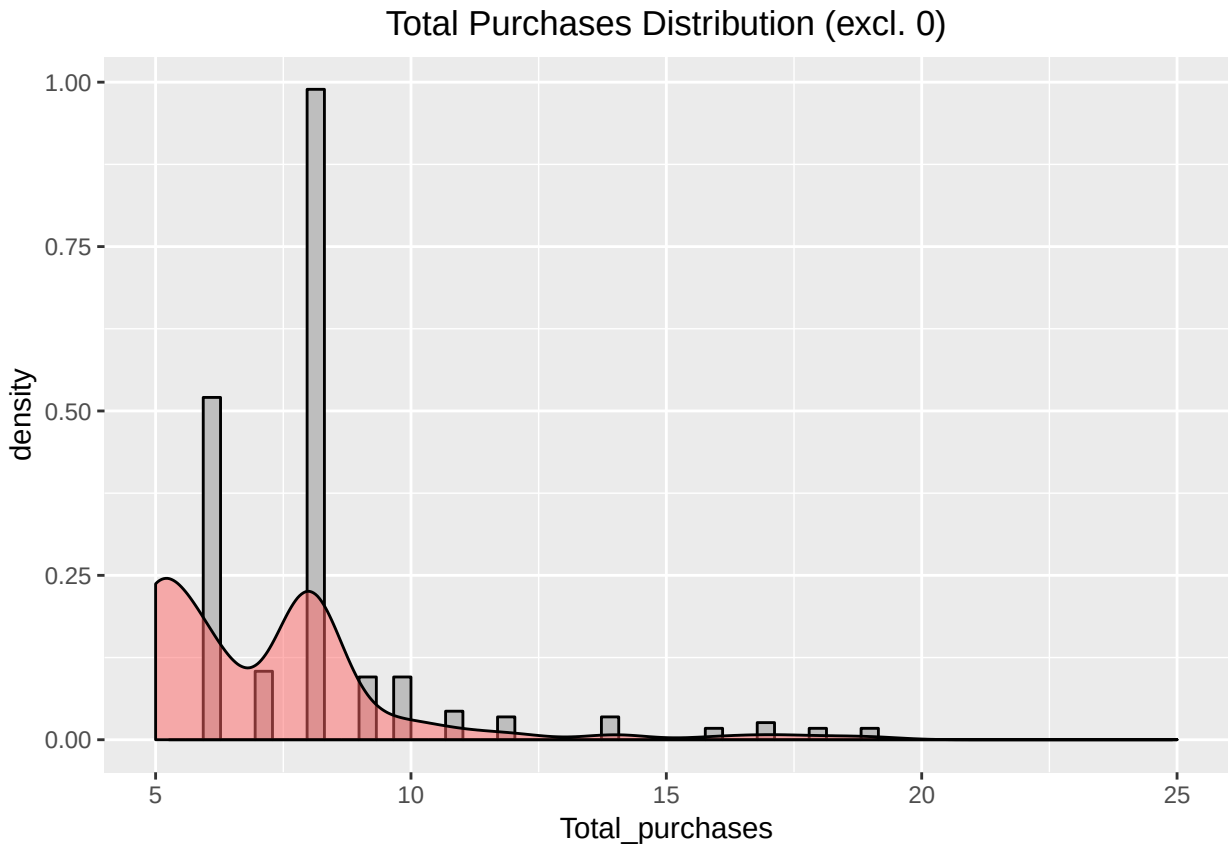
```
tphist <- ggplot(users_tot_purchase, aes(Total_purchases)) +
  geom_histogram(aes(y=..density..), colour="black", fill="gray", bins = 60) +
  xlim(5, 25) +
  geom_density(alpha=0.5, fill="#FF6666") +
  geom_vline(aes(xintercept=mean(Total_purchases)), color="blue", linetype="dashed", size=1) +
  ggtitle("Total Purchases Distribution (excl. 0)") +
  theme(plot.title = element_text(hjust=0.5))
tphist
```

```
## Warning: Removed 8680 rows containing non-finite values (stat_bin).
```

```
## Warning: Removed 8680 rows containing non-finite values (stat_density).
```

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```

```
## Warning: Removed 9020 rows containing missing values (geom_vline).
```



END PLOTTINGS

Within collaborative filtering, there are different ways to approach the problem. We will try two different methods. The first one is item-based, the second one is user-based. We will elaborate more on the differences between these two in the following paragrahs.

ITEM BASED COLLABORATIVE FILTERING (I-BCF)

First of all, we need to get rid of the data that is not helping our model. That is, the users and items that have very low quantities.

```
delete <- c()
for(i in 1:dim(train_vcuinformation)[1]){
  if(sum(train_vcuinformation [i, 11:70]) == 0){
    delete <- c(delete, i)
  }
}
train_vcuinformation <- train_vcuinformation [-delete, ]

delete_col <- c()
for(i in 11:70){
  if(sum(train_vcuinformation [, i]) == 0){
    delete_col <- c(delete_col, i)
  }
```

```
}
train_vcuinformation  <- train_vcuinformation[, -delete_col]
train_vcuinformation <- (train_vcuinformation[, !(names(train_vcuinformation) %in% c('X20', 'X25'))])

head(train_vcuinformation)
```

```
##    date    user_ID user_level plus gender age marital_status education
## 2     1 006bc1d136          4    1     -1  30              1         4
## 3     1 00e0fcf701          4    1     -1  30              1         4
## 4     1 00fbda605c          4    0     -1  30              0         3
## 7     1 020c99586c          2    1     -1  30              0         3
## 8     1 037933dc5a          4    0     -1  30              0         3
## 11    1 05791f220a          4    1     -1  30              0         4
##    city_level purchase_power X1 X2 X3 X5 X7 X8 X10 X11 X12 X13 X14 X16 X17
## 2           1              2  0  1  0  0  0  0   0   0   0   0   0   0   0
## 3           2              2  0  1  0  0  0  0   0   0   0   0   0   0   0
## 4           1              2  0  1  0  0  0  0   0   0   0   0   0   0   0
## 7           2              2  0  0  0  1  0  0   0   0   0   0   0   0   0
## 8           2              2  0  1  0  0  0  0   0   0   1   0   0   0   0
## 11          1              2  0  1  0  1  0  0   0   0   0   0   0   0   0
##    X18 X19 X21 X22 X23 X24 X26 X28 X29 X30 X31 X32 X33 X34 X36 X37 X38 X39
## 2    0   0   0   0   0   0   0   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
## 4    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 7    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 8    0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0
## 11   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##    X40 X41 X42 X43 X45 X46 X47 X48 X49 X50 X51 X52 X53 X54 X55 X56 X57 X58
## 2    0   0   0   0   0   0   0   0   0   0   0   6   0   0   0   0   0   0
## 3    0   0   0   0   0   0   0   0   0   0   0   6   0   0   0   0   0   0
## 4    0   0   0   0   0   0   0   0   0   0   0   6   0   0   0   0   0   0
## 7    0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0   0   0
## 8    0   0   0   0   0   0   0   1   0   0   0   6   0   0   0   0   0   0
## 11   0   0   0   0   0   0   0   0   0   0   0   6   0   0   0   0   0   0
##    X60
## 2    1
## 3    1
## 4    1
## 7    0
## 8    0
## 11   0
```

In item-based collaborative filtering we are not concerned about the users, so we can disregard the user column of our dataframe. We will drop it using the following code:

```
single_customer_purchase_information <- rowsum(train_vcuinformation[,11:dim(train_vcuinformation)[2]],t
names(single_customer_purchase_information) <- names(train_vcuinformation)[11:dim(train_vcuinformation)
```

The next step is to build a function that will help us determine the association relationships between different items (sku's). The function we use for this section is the Cosine Similarity. Essentially, cosine similarity takes the sum product of two attributes and divides it by the product of the square root of the sum of squares of each individual attribute. The output represents how similar an attribute is to another attribute. Values can range from [a,b] where a denotes X, and b denotes Y.

```
getCosine <- function(x,y){
  this.cosine <- sum(x*y) / (sqrt(sum(x*x)) * sqrt(sum(y*y)))
  return(this.cosine)
}
```

We are ready to compare the items, but we need to create a placeholder object to store the values of similarities. The placeholder will be filled in with values of similarities between items; hence it will be a diagonal matrix with items as rows, as well as the columns.

```
vcuinformation_similarity <- matrix(NA,nrow=ncol(single_customer_purchase_information),ncol=ncol(single_
```

Looping through the columns we compute the similarities using the helper function previously defined:

```
# Lets fill in those empty spaces with cosine similarities
# Loop through the columns
for(i in 1:ncol(single_customer_purchase_information)) {
  # Loop through the columns for each column
  for(j in 1:ncol(single_customer_purchase_information)) {
    # Fill in placeholder with cosine similarities
    vcuinformation_similarity[i,j] <- getCosine(as.matrix(single_customer_purchase_information[i]),as.ma
  }
}

# Back to dataframe
vcuinformation_similarity <- as.data.frame(vcuinformation_similarity)
```

Rephrase: Note: For loops in R are infernally slow. We use as.matrix() to transform the columns into matrices since matrix operations run a lot faster. We transform the similarity matrix into a data.frame for later processes that we will use.

Now that we have our similarity matrix, we want to make recommendations. We can do this by looking at the top K most similar skus, to each given sku.

Again we create a placeholder object, and proceed to fill it in:

```
vcuinformation_neighbours <- matrix(NA, nrow=ncol(vcuinformation_similarity),ncol=11,dimnames=list(colna
```

We loop to find the neighbouring skus:

```
for(i in 1:ncol(single_customer_purchase_information)) {
  vcuinformation_neighbours[i,] <- (t(head(n=11,rownames(vcuinformation_similarity[order(vcuinformation_
}
```

The output of I-BCF model is:

```
vcuinformation_neighbours <- vcuinformation_neighbours[, -1]
head(vcuinformation_neighbours)
```

```
##    [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]  [,10]
## X1 "X3"  "X53" "X41" "X19" "X22" "X33" "X56" "X18" "X26" "X14"
## X2 "X52" "X60" "X5"  "X48" "X43" "X46" "X37" "X38" "X28" "X51"
## X3 "X13" "X17" "X22" "X1"  "X41" "X19" "X23" "X53" "X43" "X36"
## X5 "X52" "X2"  "X48" "X60" "X16" "X38" "X14" "X24" "X29" "X1"
## X7 "X40" "X45" "X42" "X34" "X1"  "X2"  "X3"  "X5"  "X8"  "X10"
## X8 "X16" "X13" "X24" "X12" "X55" "X56" "X50" "X3"  "X19" "X60"
```

USER BASED COLLABORATIVE FILTERING (U-BCF)

The second approach we can take towards collaborative filtering is one focused on the similarities between users.

```
ID <- row.names(single_customer_purchase_information)
vcuinformation_df <- cbind(ID,single_customer_purchase_information)
head(vcuinformation_df)
```

```
##                    ID X1 X2 X3 X5 X7 X8 X10 X11 X12 X13 X14 X16 X17 X18
## 004d88db13 004d88db13  0  0  0  0  0  0   0   0   0   0   0   0   0   0
## 006bc1d136 006bc1d136  0  1  0  0  0  0   0   0   0   0   0   0   0   0
## 00e0fcf701 00e0fcf701  0  1  0  0  0  0   0   0   0   0   0   0   0   0
## 00fbda605c 00fbda605c  0  1  0  0  0  0   0   0   0   0   0   0   0   0
## 016c1ee449 016c1ee449  0  1  0  0  0  0   0   0   0   0   0   0   0   0
## 02033e9b30 02033e9b30  0  1  0  1  0  0   0   0   0   0   0   0   0   0
##            X19 X21 X22 X23 X24 X26 X28 X29 X30 X31 X32 X33 X34 X36 X37 X38
## 004d88db13   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2
## 006bc1d136   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 00e0fcf701   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 00fbda605c   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 016c1ee449   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 02033e9b30   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##            X39 X40 X41 X42 X43 X45 X46 X47 X48 X49 X50 X51 X52 X53 X54 X55
## 004d88db13   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 006bc1d136   0   0   0   0   0   0   0   0   0   0   0   0   6   0   0   0
## 00e0fcf701   0   0   0   0   0   0   0   0   0   0   0   0   6   0   0   0
## 00fbda605c   0   0   0   0   0   0   0   0   0   0   0   0   6   0   0   0
## 016c1ee449   0   0   0   0   0   0   0   0   0   0   0   0   3   0   0   0
## 02033e9b30   0   0   0   0   0   0   0   0   0   0   0   0   3   0   0   0
##            X56 X57 X58 X60
## 004d88db13   0   0   0   0
## 006bc1d136   0   0   0   1
## 00e0fcf701   0   0   0   1
## 00fbda605c   0   0   0   1
## 016c1ee449   0   0   0   1
## 02033e9b30   0   0   0   1
```

For this approach, we want to construct a score matrix, and we will follow a very similar process to the one in I-BCF. The steps are as follows: -Pick an sku, and check whether a given customer has purchased the item -Get that item's top K neighbours similarities -Get the consumption record of customer for those K neighbour items. -Compute the relative score for the user and those neighbouring items by means of: sumproduct(purchaseHistory, similarities)/sum(similarities)

Starting with the score computation part, we create a new helper function for this calculation:

```
# Lets make a helper function to calculate the scores
getScore <- function(history, similarities) {
  x <- sum(history*similarities)/sum(similarities)
  x
}
```

Placeholder matrix follows:

```
holder <- matrix(NA, nrow=nrow(vcuinformation_df),ncol=ncol(vcuinformation_df)-1,dimnames=list((vcuinfo
dim(holder) # row-users, col-items
```

```
## [1] 412  50
```

Next, we must loop through the dataframe and calculate the quantities that will be inputs to our score function.

Rephrase: The loop starts by taking each user (row) and then jumps into another loop that takes each column (artists). We then store the user's name and artist name in variables to use them easily later. We then use an if statement to filter out artists that a user has already listened to – this is a business case decision.

```
# Loop through the IDs (rows)
   for(i in 1:nrow(holder))
   {
       # Loops through the products (columns)
       for(j in 1:ncol(holder))
       {
           # Get the ID's name and th product's name
           # We do this not to conform with vectors sorted differently
             ID <- rownames(holder)[i]
             product <- colnames(holder)[j]

           # We do not want to recommend products you have already consumed
           # If you have already consumed it, we store an empty string
             if(as.integer(vcuinformation_df[vcuinformation_df$ID==ID,product]) == 1)
             {
                 holder[i,j]<-""
              } else {

           # We first have to get a product's top 10 neighbours sorted by similarity
             topN<-((head(n=11,(vcuinformation_similarity[order(vcuinformation_similarity[,product],dec
             topN.names <- as.character(rownames(topN))
             topN.similarities <- as.numeric(topN[,1])

           # Drop the first one because it will always be the same song
             topN.similarities<-topN.similarities[-1]
             topN.names<-topN.names[-1]

           # We then get the ID's purchase history for those 10 items
             topN.purchases<- vcuinformation_df[,c("ID",topN.names)]
             topN.IDPurchases <-topN.purchases[topN.purchases$ID==ID,]
             topN.IDPurchases <- as.numeric(topN.IDPurchases[!(names(topN.IDPurchases) %in% c("ID"))])

            # We then calculate the score for that product and that ID
             holder[i,j]<-getScore(similarities=topN.similarities, history=topN.IDPurchases)

         } # close else statement
       } # end product for loop
   } # end ID for loop

vcuinformation_df_ID_scores <- holder
head(vcuinformation_df_ID_scores)
```

```
##            X1  X2                  X3  X5                  X7
```

```
## 004d88db13 "0" "0.0374447007661357" "0" "0.0697620013352452" "0"
## 006bc1d136 "0" ""                   "0" "2.78372669390051"   "0"
## 00e0fcf701 "0" ""                   "0" "2.78372669390051"   "0"
## 00fbda605c "0" ""                   "0" "2.78372669390051"   "0"
## 016c1ee449 "0" ""                   "0" "1.58737120020929"   "0"
## 02033e9b30 "0" ""                   "0" ""                   "0"
##            X8                    X10 X11 X12                  X13
## 004d88db13 "0"                    "0" "0" "0"                  "0"
## 006bc1d136 "0.00240223576899755" "0" "0" "0.0901222271010641" "0"
## 00e0fcf701 "0.00240223576899755" "0" "0" "0.0901222271010641" "0"
## 00fbda605c "0.00240223576899755" "0" "0" "0.0901222271010641" "0"
## 016c1ee449 "0.00240223576899755" "0" "0" "0.058163945423155"  "0"
## 02033e9b30 "0.00240223576899755" "0" "0" "0.058163945423155"  "0"
##            X14                   X16                  X17 X18 X19 X21 X22
## 004d88db13 "0"                   "0"                  "0" "0" "0" "0" "0"
## 006bc1d136 "0.0304968986788702" "0.0722670051143906" "0" "0" "0" "0" "0"
## 00e0fcf701 "0.0304968986788702" "0.0722670051143906" "0" "0" "0" "0" "0"
## 00fbda605c "0.0304968986788702" "0.0722670051143906" "0" "0" "0" "0" "0"
## 016c1ee449 "0.0304968986788702" "0.0361335025571953" "0" "0" "0" "0" "0"
## 02033e9b30 "0.0304968986788702" "0.0886158890552792" "0" "0" "0" "0" "0"
##            X23                   X24                  X26
## 004d88db13 "0"                   "0"                  "0"
## 006bc1d136 "0.0838241758048648" "0.0722595144607333" "0"
## 00e0fcf701 "0.0838241758048648" "0.0722595144607333" "0"
## 00fbda605c "0.0838241758048648" "0.0722595144607333" "0"
## 016c1ee449 "0.0577635626991321" "0.0446165987125976" "0"
## 02033e9b30 "0.0577635626991321" "0.0560880752059649" "0"
##            X28                  X29                X30 X31 X32
## 004d88db13 "0"                  "0"                "0" "0" "0"
## 006bc1d136 "0.312060485098866" "1.46172093447317" "0" "0" "0"
## 00e0fcf701 "0.312060485098866" "1.46172093447317" "0" "0" "0"
## 00fbda605c "0.312060485098866" "1.46172093447317" "0" "0" "0"
## 016c1ee449 "0.208976418171829" "1.0712639044814"  "0" "0" "0"
## 02033e9b30 "0.208976418171829" "1.26030468666118" "0" "0" "0"
##            X33                  X34 X36 X37
## 004d88db13 "0"                  "0" "0" "0"
## 006bc1d136 "0.282417909517546" "0" "0" "0.14995740233703"
## 00e0fcf701 "0.282417909517546" "0" "0" "0.14995740233703"
## 00fbda605c "0.282417909517546" "0" "0" "0.14995740233703"
## 016c1ee449 "0.180477883582943" "0" "0" "0.14995740233703"
## 02033e9b30 "0.180477883582943" "0" "0" "0.14995740233703"
##            X38                  X39 X40 X41 X42 X43                  X45
## 004d88db13 "0"                  "0" "0" "0" "0" "0"                  "0"
## 006bc1d136 "0.596716041955252" "0" "0" "0" "0" "0.825429874971247" "0"
## 00e0fcf701 "0.596716041955252" "0" "0" "0" "0" "0.825429874971247" "0"
## 00fbda605c "0.596716041955252" "0" "0" "0" "0" "0.825429874971247" "0"
## 016c1ee449 "0.596716041955252" "0" "0" "0" "0" "0.493479104480851" "0"
## 02033e9b30 "1"                 "0" "0" "0" "0" "0.493479104480851" "0"
##            X46                   X47                  X48
## 004d88db13 "0"                   "0"                  "0"
## 006bc1d136 "0.0268092401323755" "0.0736046737300039" "2.55268387710996"
## 00e0fcf701 "0.0268092401323755" "0.0736046737300039" "2.55268387710996"
## 00fbda605c "0.0268092401323755" "0.0736046737300039" "2.55268387710996"
## 016c1ee449 "0.0268092401323755" "0.0430062159983974" "1.45567888635812"
```

```
## 02033e9b30 "0.0268092401323755" "0.0430062159983974" "1.55631173400503"
##            X49 X50 X51                 X52                 X53 X54
## 004d88db13 "0" "0" "0"                 "0"                 "0" "0"
## 006bc1d136 "0" "0" "0.340992318855734" "0.527136079912527" "0" "0"
## 00e0fcf701 "0" "0" "0.340992318855734" "0.527136079912527" "0" "0"
## 00fbda605c "0" "0" "0.340992318855734" "0.527136079912527" "0" "0"
## 016c1ee449 "0" "0" "0.199237202956668" "0.527136079912527" "0" "0"
## 02033e9b30 "0" "0" "0.199237202956668" "0.749419669692106" "0" "0"
##            X55                  X56 X57 X58
## 004d88db13 "0"                  "0" "0" "0"
## 006bc1d136 "0.116970819668047"  "0" "0" "2.07537875881079"
## 00e0fcf701 "0.116970819668047"  "0" "0" "2.07537875881079"
## 00fbda605c "0.116970819668047"  "0" "0" "2.07537875881079"
## 016c1ee449 "0.0584854098340236" "0" "0" "1.43015150352432"
## 02033e9b30 "0.0584854098340236" "0" "0" "1.43015150352432"
##            X60
## 004d88db13 "0.0404009727204225"
## 006bc1d136 ""
## 00e0fcf701 ""
## 00fbda605c ""
## 016c1ee449 ""
## 02033e9b30 ""
```

```
dt = as.data.frame(vcuinformation_df_ID_scores)
user_ID = row.names(vcuinformation_df_ID_scores)
dt = cbind(user_ID, dt)
# dt

# vcuinformation_df_ID_scores[1:5,]
# write.csv(vcuinformation_df_ID_scores[1:5,],"/Users/stefanozavagli/Documents/Carrera/Schools/Berkeley/
```

We can now get the item-based similarity score for each sku under consideration:

The number K of neighbours that we pick will influence significantly the end results. When storing the values, the first column can be dropped if we do not wish to recommend the same item again the the customer. This is however a business case choice.

We just need the user's purchase history for the top 10 songs.

We use the original data set to get the purchases of our users' top 10 purchases.

We filter out our current user in the loop and then filter out purchases that match the user.

We are now ready to calculate the score and store it in our holder matrix:

This basically reads that for user 51 we would recommend abba first, then a perfect circle, and we would not recommend ACDC.

This is not very pretty . . . so lets make it pretty:

We will create another holder matrix and for each user score we will sort the scores and store the artist names in rank order.

```
single_customer_purchase_information_test <- rowsum(test_vcuinformation[,11:70],test_vcuinformation$use
# single_customer_purchase_information_test
```

```
accuracy_function <- function(n,vcuinformation_df_ID_scores_holder,single_customer_purchase_information
    cpt <- 0
  for(i in 1:nrow(single_customer_purchase_information)){
    if (sum(single_customer_purchase_information[i,])==0){
      cpt <- cpt + 1
    }
  }
  index <- row.names(vcuinformation_df_ID_scores_holder)
  corr <- 0
  for (i in index) {
    if (sum(single_customer_purchase_information_test[i,])>0){
    for (j in 1:n){
      produ <- vcuinformation_df_ID_scores_holder[i,j]
      if (single_customer_purchase_information[i,produ]> 0) {
        corr <- corr + 1
        break
      }
    }
    }
  }
  # print(corr)
  # print(nrow(vcuinformation_df_ID_scores_holder)-corr-cpt)
  ACC = 100*corr/(nrow(vcuinformation_df_ID_scores_holder)-cpt)
```

```
  return(ACC)
}
```

rowsum(x, group, reorder = TRUE, . . .)

```
cpt <- 0
for(i in 1:nrow(single_customer_purchase_information_test)){
  if (sum(single_customer_purchase_information_test[i,])==0){
    cpt <- cpt + 1
  }
}
cpt
```

```
## [1] 361
```

```
N = 5
ACC = c()
for (i in 1:N) {
  number_recommandation = i   # Number of recommended product you want to obtain
  # Lets make our recommendations pretty
  vcuinformation_df_ID_scores_holder <- matrix(NA, nrow=nrow(vcuinformation_df_ID_scores),ncol=number_r
  for(j in 1:nrow(vcuinformation_df_ID_scores)) {
    vcuinformation_df_ID_scores_holder[j,] <- names(head(n=number_recommandation,(vcuinformation_df_ID_s
  }


  ACC[i] = accuracy_function(i,vcuinformation_df_ID_scores_holder,single_customer_purchase_information_
  if (i <= 5){
    print(head(vcuinformation_df_ID_scores_holder))
  }
  print(ACC[i])
}
```

```
##            [,1]
## 004d88db13 "X5"
## 006bc1d136 "X5"
## 00e0fcf701 "X5"
## 00fbda605c "X5"
## 016c1ee449 "X5"
## 02033e9b30 "X48"
## [1] 21.56863
##            [,1]  [,2]
## 004d88db13 "X5"  "X60"
## 006bc1d136 "X5"  "X48"
## 00e0fcf701 "X5"  "X48"
## 00fbda605c "X5"  "X48"
## 016c1ee449 "X5"  "X48"
## 02033e9b30 "X48" "X58"
## [1] 25.4902
##            [,1]  [,2]  [,3]
## 004d88db13 "X5"  "X60" "X2"
## 006bc1d136 "X5"  "X48" "X58"
## 00e0fcf701 "X5"  "X48" "X58"
## 00fbda605c "X5"  "X48" "X58"
## 016c1ee449 "X5"  "X48" "X58"
```

```
## 02033e9b30 "X48" "X58" "X29"
## [1] 27.45098
##            [,1]  [,2]  [,3]  [,4]
## 004d88db13 "X5"  "X60" "X2"  "X1"
## 006bc1d136 "X5"  "X48" "X58" "X29"
## 00e0fcf701 "X5"  "X48" "X58" "X29"
## 00fbda605c "X5"  "X48" "X58" "X29"
## 016c1ee449 "X5"  "X48" "X58" "X29"
## 02033e9b30 "X48" "X58" "X29" "X38"
## [1] 29.41176
##            [,1]  [,2]  [,3]  [,4]  [,5]
## 004d88db13 "X5"  "X60" "X2"  "X1"  "X3"
## 006bc1d136 "X5"  "X48" "X58" "X29" "X43"
## 00e0fcf701 "X5"  "X48" "X58" "X29" "X43"
## 00fbda605c "X5"  "X48" "X58" "X29" "X43"
## 016c1ee449 "X5"  "X48" "X58" "X29" "X38"
## 02033e9b30 "X48" "X58" "X29" "X38" "X52"
## [1] 31.37255
```

```
dt = as.data.frame(vcuinformation_df_ID_scores_holder)
user_ID = row.names(vcuinformation_df_ID_scores_holder)
dt = cbind(user_ID, dt)
head(dt)
```

```
##               user_ID  V1  V2  V3  V4  V5
## 004d88db13 004d88db13  X5 X60  X2  X1  X3
## 006bc1d136 006bc1d136  X5 X48 X58 X29 X43
## 00e0fcf701 00e0fcf701  X5 X48 X58 X29 X43
## 00fbda605c 00fbda605c  X5 X48 X58 X29 X43
## 016c1ee449 016c1ee449  X5 X48 X58 X29 X38
## 02033e9b30 02033e9b30 X48 X58 X29 X38 X52
```

```
sprintf("Percent Accuracy for %s recommendations is: %s", i, ACC[i])
```

```
## [1] "Percent Accuracy for 5 recommendations is: 31.3725490196078"
```

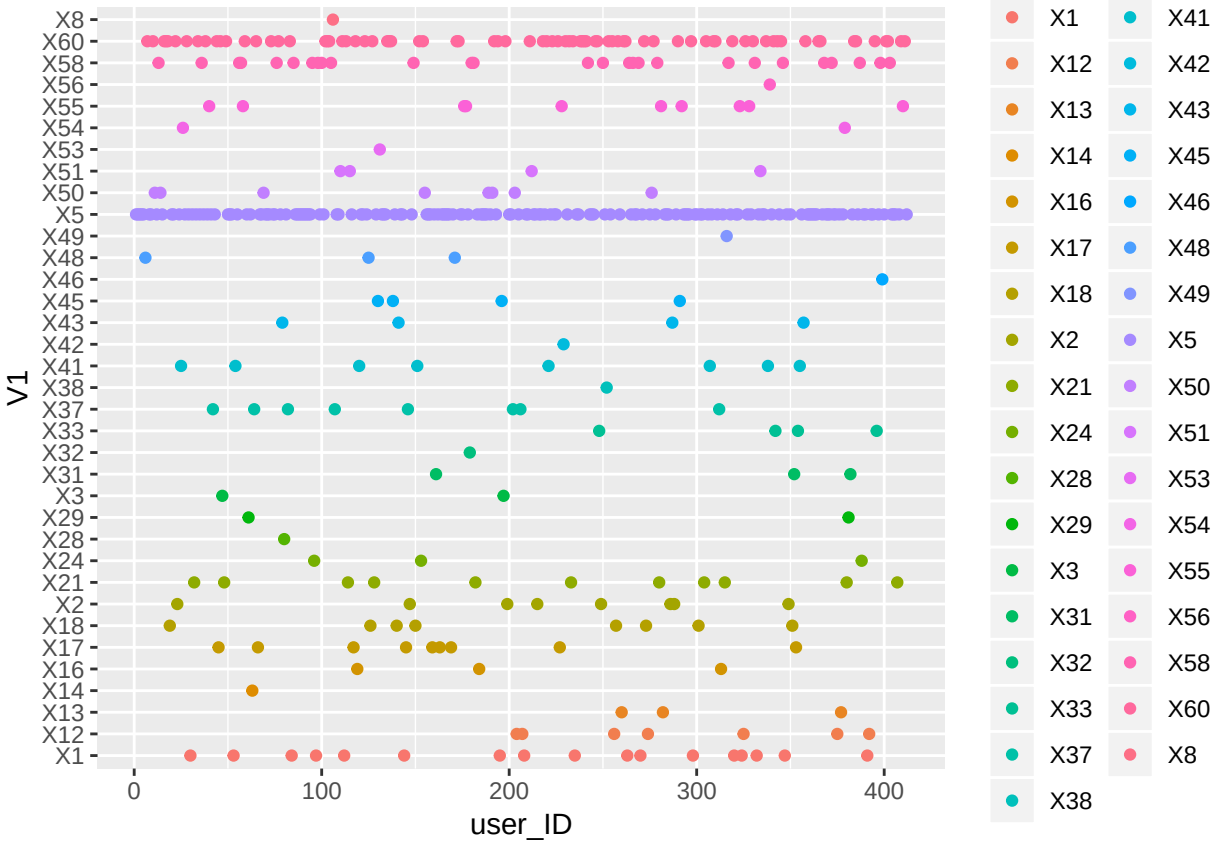
```
xaxis <- seq(1,50)
ACC
```

```
## [1] 21.56863 25.49020 27.45098 29.41176 31.37255
```

```
dat <- as.data.frame(xaxis)
dat[2] <- as.data.frame(ACC)
names(dat)[1] = "xaxis"
names(dat)[2] = "ACC"

benchmark<- as.data.frame(seq(1,50))
benchmark[2] <- as.data.frame(seq(from = 2, to = 100, by = 2))
names(benchmark)[1] = "x"
names(benchmark)[2] = "AC"

accuracy_plot = ggplot(data = dat, aes(x = xaxis, y = ACC)) +
  geom_point(aes(color = ACC)) +
  geom_line(data = benchmark, aes(x = x, y = AC), color = "red",linetype='dashed') +
  xlab("# recommendations") +
  ylab("Accuracy [%]") +
  ggtitle("Accuracy vs Number of Recommendations") +
```

```
  theme(plot.title = element_text(hjust=0.5))
accuracy_plot
```


```
# Scatter Plot

# transform vcuinformation_df_ID_scores_holder into continuous data
df = as.data.frame(vcuinformation_df_ID_scores_holder)
user_ID = row.names(vcuinformation_df_ID_scores_holder)
dat = cbind(user_ID, df)
dat$user_ID = as.numeric(seq(1, length(dat$user_ID)))

dat_del = dat
for(i in 2:dim(dat_del)[2]) {
  dat_del[,i] = as.character(dat_del[,i])
  dat_del[,i]  = sub(".", "", dat_del[,i])
  dat_del[,i]  = as.integer(dat_del[,i])
}

# colored scatterplot
ggplot(data = dat, aes(x = user_ID, y = V1)) +
  geom_point(aes(color = V1))
```



ENSEMBLE MODELS: CF + LR

Merge 2 dataframes by user_id. That is, add these three columns to the original dataset

```
vcuinformation_df_ID_scores_holder = as.data.frame(vcuinformation_df_ID_scores_holder)
user_ID = row.names(vcuinformation_df_ID_scores_holder)
vcuinformation_df = cbind(user_ID, vcuinformation_df_ID_scores_holder)

vcuinformation_df_del = vcuinformation_df
for(i in 2:dim(vcuinformation_df)[2]) {
  vcuinformation_df_del[,i] = as.character(vcuinformation_df_del[,i])
  vcuinformation_df_del[,i]  = sub(".", "", vcuinformation_df_del[,i])
  vcuinformation_df_del[,i]  = as.integer(vcuinformation_df_del[,i])
}

train_vcui_merge = merge(train_vcuinformation, vcuinformation_df_del, by = c("user_ID"))

train_vcui_mod = train_vcui_merge[1:(dim(train_vcui_merge)[2])]
train_vcui_mod$user_ID = as.numeric(seq(1, length(train_vcuinformation$user_ID)))
# train_vcui_mod
# train_vcui_mod$V1 = as.factor(train_vcui_mod$V1)
# train_vcui_mod$V1 = relevel(train_vcui_mod$V1, ref="X20")
```

```r
model = lm(user_ID ~ . - date, data = train_vcui_mod)
summary(model)
```

```
##
## Call:
## lm(formula = user_ID ~ . - date, data = train_vcui_mod)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -330.85 -109.03    0.69  114.73  334.73
##
## Coefficients: (1 not defined because of singularities)
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)     383.6659    82.5555   4.647 4.37e-06 ***
## user_level       -2.2155     9.9241  -0.223   0.8234
## plus             -8.9517    16.4606  -0.544   0.5868
## gender            0.3651     8.9807   0.041   0.9676
## age              -1.5013     0.9102  -1.649   0.0997 .
## marital_status   27.9899    16.6406   1.682   0.0932 .
## education        -7.2710    14.9069  -0.488   0.6259
## city_level        5.8333     6.6863   0.872   0.3834
## purchase_power  -20.3549    17.2336  -1.181   0.2382
## X1               31.4841    29.4408   1.069   0.2854
## X2               13.2488    11.8945   1.114   0.2659
## X3               18.2482    25.2745   0.722   0.4707
## X5               59.0059    29.5359   1.998   0.0463 *
## X7              235.7492   267.9956   0.880   0.3795
## X8               -0.3792     6.8199  -0.056   0.9557
## X10              27.9855    31.4596   0.890   0.3742
## X11              -7.2493   106.6293  -0.068   0.9458
## X12              -1.4249    32.4376  -0.044   0.9650
## X13             -57.1621    48.8203  -1.171   0.2422
## X14             -61.1095    57.2532  -1.067   0.2864
## X16            -111.1496   106.5801  -1.043   0.2975
## X17               5.6024    82.2178   0.068   0.9457
## X18               2.0309     5.9206   0.343   0.7317
## X19              12.3331    16.8622   0.731   0.4649
## X21             218.4183   160.7747   1.359   0.1749
## X22             -36.9817    27.4919  -1.345   0.1792
## X23             -78.8146    91.3117  -0.863   0.3885
## X24              37.4339    26.5848   1.408   0.1598
## X26              41.5296    20.1821   2.058   0.0402 *
## X28              13.8887   123.3983   0.113   0.9104
## X29             -11.4124    12.9360  -0.882   0.3781
## X30              -1.0000   216.3727  -0.005   0.9963
## X31            -166.9258   157.9113  -1.057   0.2910
## X32             -47.6889   156.4849  -0.305   0.7607
## X33               5.2804    13.4255   0.393   0.6943
## X34               8.9442     4.2740   2.093   0.0369 *
## X36             -26.5305    45.7261  -0.580   0.5621
## X37              -6.2076     8.5047  -0.730   0.4658
## X38            -171.3841    94.1003  -1.821   0.0692 .
## X39             -51.1547    82.9316  -0.617   0.5376
```

```
## X40                  NA         NA      NA       NA
## X41            144.9413    63.7346   2.274   0.0234 *
## X42             17.8458    21.9698   0.812   0.4170
## X43           -146.6626   112.1390  -1.308   0.1916
## X45              7.7904   218.6997   0.036   0.9716
## X46              5.1193   101.1818   0.051   0.9597
## X47             38.7110    30.3930   1.274   0.2034
## X48             15.2100    24.6956   0.616   0.5383
## X49              3.5658   166.3494   0.021   0.9829
## X50              2.5956     3.8800   0.669   0.5038
## X51             21.8581   123.2722   0.177   0.8593
## X52             -1.5724     4.5165  -0.348   0.7279
## X53            -21.2694    14.2982  -1.488   0.1375
## X54             -8.7014    20.8539  -0.417   0.6767
## X55            -19.2371    20.3344  -0.946   0.3446
## X56            -19.3965    11.9444  -1.624   0.1051
## X57             55.2320   114.0045   0.484   0.6283
## X58           -146.1323   111.3680  -1.312   0.1901
## X60             22.4527    17.0089   1.320   0.1875
## V1              -0.5068     0.5240  -0.967   0.3340
## V2              -0.1381     0.5333  -0.259   0.7957
## V3              -0.3877     0.5109  -0.759   0.4483
## V4               0.3843     0.5048   0.761   0.4468
## V5              -0.6598     0.6612  -0.998   0.3188
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 153 on 470 degrees of freedom
## Multiple R-squared:  0.1281, Adjusted R-squared:  0.01307
## F-statistic: 1.114 on 62 and 470 DF,  p-value: 0.268
```

```
# vif(model)
# Anova(model)
print("R^2 is 11.8%")
```

```
## [1] "R^2 is 11.8%"
```

```
test_vcui_mod = test_vcuinformation[(1+1):(dim(test_vcuinformation)[2]-1)]
test_vcui_mod$user_ID = as.numeric((seq(1, length(test_vcuinformation$user_ID))))

# We can't give a prediction for user that we haven't estimated in the training.
# pred = predict(model, newdata=test_vcui_mod)
# SSE = sum((test_vcui_mod$user_ID - pred)^2)
# SST = sum((test_vcui_mod$user_ID - mean(train_vcui_mod$user_ID))^2)
# OSR2 = 1 - SSE/SST
# OSR2
```

```
# train_vcui_mod
```

```
# test_vcui_mod
```

Merge 2 dataframes by user_id. That is, add these three columns to the original dataset

```
vcuinformation_df_ID_scores_holder = as.data.frame(vcuinformation_df_ID_scores_holder)
user_ID = row.names(vcuinformation_df_ID_scores_holder)
vcuinformation_df = cbind(user_ID, vcuinformation_df_ID_scores_holder)

train_vcui_merge = merge(train_vcuinformation, vcuinformation_df, by = c("user_ID"))

train_vcui_mod = train_vcui_merge[1:(dim(train_vcui_merge)[2]-3)]
train_vcui_mod$user_ID = as.numeric(seq(1, length(train_vcuinformation$user_ID)))
# train_vcui_mod

user_ID <- train_vcui_mod[,1]
j <- train_vcui_mod[,3:length(colnames(train_vcui_mod))]
train_vcui_mod <- cbind(user_ID,j)

# train_vcui_mod

# train_vcui_mod$V1 = as.factor(train_vcui_mod$V1)
# train_vcui_mod$V1 = relevel(train_vcui_mod$V1, ref="X20")

model = lm(user_ID ~ ., data = train_vcui_mod)
summary(model)
```

```
##
## Call:
## lm(formula = user_ID ~ ., data = train_vcui_mod)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -287.90  -83.54    0.00   90.80  371.44
##
## Coefficients: (8 not defined because of singularities)
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     401.6079   133.0475   3.019   0.0027 **
## user_level       -2.1737    10.8768  -0.200   0.8417
## plus             -2.3153    17.5143  -0.132   0.8949
## gender            1.9823     9.9677   0.199   0.8425
## age              -0.7751     1.0029  -0.773   0.4400
## marital_status   26.2910    17.5239   1.500   0.1343
## education        -1.2656    16.7251  -0.076   0.9397
## city_level        8.4136     8.2314   1.022   0.3073
## purchase_power   -6.3417    18.6683  -0.340   0.7343
## X1               33.2883    42.8974   0.776   0.4382
## X2               20.9277    14.1300   1.481   0.1394
## X3              -33.9538    49.7572  -0.682   0.4954
## X5               90.3839    43.5648   2.075   0.0386 *
## X7              184.0626   414.3227   0.444   0.6571
## X8               -0.5224    10.7263  -0.049   0.9612
## X10             -36.4844    46.6907  -0.781   0.4350
## X11            -737.5456   560.5446  -1.316   0.1890
## X12             -69.3959    57.4095  -1.209   0.2274
```

```
## X13            -25.7963    63.0051  -0.409   0.6824
## X14            -89.4587    99.4620  -0.899   0.3690
## X16             21.3333   137.7224   0.155   0.8770
## X17             -8.6530   114.6830  -0.075   0.9399
## X18              2.3199     7.9656   0.291   0.7710
## X19             32.2056    30.5800   1.053   0.2929
## X21            -54.8485   187.1320  -0.293   0.7696
## X22             23.9434    39.6909   0.603   0.5467
## X23           -125.9904   108.1843  -1.165   0.2449
## X24             39.3905    44.3515   0.888   0.3750
## X26             19.0935    23.8628   0.800   0.4241
## X28             27.7823   134.4701   0.207   0.8364
## X29             -0.7974    13.2854  -0.060   0.9522
## X30             -1.0000   208.2512  -0.005   0.9962
## X31            -26.6033   280.6091  -0.095   0.9245
## X32           -493.7284   646.6386  -0.764   0.4456
## X33            -11.3829    19.6353  -0.580   0.5624
## X34             -1.0062     6.5814  -0.153   0.8786
## X36             21.4119    74.4022   0.288   0.7737
## X37              1.5416    11.3545   0.136   0.8921
## X38           -169.3517    92.6108  -1.829   0.0682 .
## X39             11.1657   106.6105   0.105   0.9166
## X40                  NA         NA      NA       NA
## X41             32.7282    98.5815   0.332   0.7401
## X42             33.2714    28.4502   1.169   0.2429
## X43           -119.2189   111.8607  -1.066   0.2872
## X45           -104.8887   365.4408  -0.287   0.7742
## X46             77.4862   129.7047   0.597   0.5506
## X47            -61.3633   137.4246  -0.447   0.6555
## X48             -2.3199    30.5641  -0.076   0.9395
## X49            982.3628   596.3835   1.647   0.1003
## X50             -1.7441     4.4837  -0.389   0.6975
## X51             14.7737   204.5977   0.072   0.9425
## X52              2.1855     5.9284   0.369   0.7126
## X53            -12.3769    14.5696  -0.850   0.3961
## X54           -184.6364   130.1061  -1.419   0.1566
## X55             -1.5412    29.2294  -0.053   0.9580
## X56             -8.3751    20.6779  -0.405   0.6857
## X57            -27.6033   280.6091  -0.098   0.9217
## X58           -126.7537   114.6765  -1.105   0.2697
## X60             -0.8345    29.0359  -0.029   0.9771
## V1X12           -9.2393   294.5866  -0.031   0.9750
## V1X13          240.5666   144.8647   1.661   0.0976 .
## V1X14         -348.0819   220.6382  -1.578   0.1154
## V1X16         -188.4489   133.3994  -1.413   0.1585
## V1X17         -187.8196   119.5853  -1.571   0.1170
## V1X18          -34.9232    92.9788  -0.376   0.7074
## V1X2             8.7681   101.3802   0.086   0.9311
## V1X21           -1.4018   102.9055  -0.014   0.9891
## V1X24           86.0485   177.6225   0.484   0.6283
## V1X28         -184.3399   177.6531  -1.038   0.3000
## V1X29          101.3262   146.7631   0.690   0.4903
## V1X3          -227.0057   173.0179  -1.312   0.1902
## V1X31          140.9393   166.5112   0.846   0.3978
```

```
## V1X32           36.0755   790.8326   0.046   0.9636
## V1X33           60.7948   139.5168   0.436   0.6632
## V1X37         -170.7354   119.0985  -1.434   0.1525
## V1X38          -21.3627   156.2312  -0.137   0.8913
## V1X41         -129.5804   146.9210  -0.882   0.3783
## V1X42          -44.8344   171.4919  -0.261   0.7939
## V1X43         -373.8544   199.6919  -1.872   0.0619 .
## V1X45         -203.4565   362.4157  -0.561   0.5748
## V1X46                NA         NA      NA       NA
## V1X48         -270.6145   125.0251  -2.164   0.0310 *
## V1X49          913.5812   728.9692   1.253   0.2108
## V1X5            12.3285    86.3010   0.143   0.8865
## V1X50         -138.2448   279.8814  -0.494   0.6216
## V1X51          198.4800   732.8764   0.271   0.7867
## V1X53         -152.9019   168.9755  -0.905   0.3661
## V1X54         -378.1590   236.6393  -1.598   0.1108
## V1X55          -78.7212   190.2970  -0.414   0.6793
## V1X56          195.3393   222.0740   0.880   0.3796
## V1X58          -24.4481    88.4626  -0.276   0.7824
## V1X60          -88.7134    88.4451  -1.003   0.3164
## V1X8          -514.0510   373.8393  -1.375   0.1699
## V2X11          490.4985   934.5663   0.525   0.6000
## V2X12           43.6104   171.0600   0.255   0.7989
## V2X13           85.6586   125.7652   0.681   0.4962
## V2X14                NA         NA      NA       NA
## V2X16           -9.1790   276.6822  -0.033   0.9736
## V2X17         -104.2714   117.6764  -0.886   0.3761
## V2X18          -88.2338   111.0477  -0.795   0.4273
## V2X19         -235.9175    94.3655  -2.500   0.0128 *
## V2X2           -91.7922   120.6755  -0.761   0.4473
## V2X21          -93.6796   251.6573  -0.372   0.7099
## V2X22          176.2043   217.4912   0.810   0.4183
## V2X23           57.9008   116.6023   0.497   0.6198
## V2X24          136.2533   307.2019   0.444   0.6576
## V2X28          326.5282   155.9288   2.094   0.0369 *
## V2X29          -52.3767   131.8969  -0.397   0.6915
## V2X3                 NA         NA      NA       NA
## V2X30                NA         NA      NA       NA
## V2X31          -41.1344    75.9028  -0.542   0.5882
## V2X33         -100.4587   103.5247  -0.970   0.3324
## V2X36         -289.1902   246.1787  -1.175   0.2408
## V2X37         -329.8859   150.2954  -2.195   0.0287 *
## V2X38         -187.8923   135.1490  -1.390   0.1652
## V2X39         -154.4750   199.9390  -0.773   0.4402
## V2X41         -201.0407   138.2458  -1.454   0.1466
## V2X43           39.0071   118.9254   0.328   0.7431
## V2X46          -18.4385   324.4066  -0.057   0.9547
## V2X47                NA         NA      NA       NA
## V2X48         -160.3426   124.8114  -1.285   0.1996
## V2X5          -145.5829   131.6126  -1.106   0.2693
## V2X50                NA         NA      NA       NA
## V2X53           71.7533   202.2279   0.355   0.7229
## V2X54          295.4130   902.7635   0.327   0.7437
## V2X55         -267.1643   240.3293  -1.112   0.2669
```

```
## V2X56           163.0045   188.7478   0.864   0.3883
## V2X58          -103.5230   126.0987  -0.821   0.4121
## V2X60          -160.9363   124.2775  -1.295   0.1961
## V2X7                 NA         NA      NA       NA
## V2X8            34.3463   156.0068   0.220   0.8259
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 147.3 on 410 degrees of freedom
## Multiple R-squared:  0.2954, Adjusted R-squared:  0.08577
## F-statistic: 1.409 on 122 and 410 DF,  p-value: 0.007234
```

```
# vif(model)
# Anova(model)
print("R^2 is 11.8%")
```

```
## [1] "R^2 is 11.8%"
```

```
test_vcui_mod = test_vcuinformation[(1+1):(dim(test_vcuinformation)[2]-1)]
test_vcui_mod$user_ID = as.numeric((seq(1, length(test_vcuinformation$user_ID))))
```

```
a <- test_vcui_mod[,1:12]
X5 <- test_vcui_mod[,14]
a <- cbind(a,X5)
a <- cbind(a,test_vcui_mod[,16:17])
a <- cbind(a,test_vcui_mod[,19:23])
a <- cbind(a,test_vcui_mod[,26:29])
a <- cbind(a,test_vcui_mod[,31:35])
a <- cbind(a,test_vcui_mod[,37:43])
a <- cbind(a,test_vcui_mod[,45:52])
a <- cbind(a,test_vcui_mod[,54:67])
X60 <- test_vcui_mod[,69]
a <- cbind(a,X60)

# a
```

```
# We can't give a prediction for user that we haven't estimated in the training.
# pred = predict(model, newdata=a)
# SSE = sum((a$user_ID - pred)^2)
# SST = sum((a$user_ID - mean(a$user_ID))^2)
# OSR2 = 1 - SSE/SST
# OSR2
```

Appendix - Linear Regression

```
setwd("/Users/stefanozavagli/Documents/Carrera/Schools/Berkeley/Clases/SPRING '20/IEOR 242/Project/Data/")

train_skui = read.csv("train_skuinformation_missingvalue_removed.csv")
# train_skui = read.csv("train_skuinformation.csv")
train_vcui = read.csv("train_vcuinformation.csv")
test_skui = read.csv("test_skuinformation_missingvalue_removed.csv")
# train_skui = read.csv("test_skuinformation.csv")
test_vcui = read.csv("test_vcuinformation.csv")
```

```
# plot(train_skui)
# plot(test_skui)

head(train_skui)
```

```
##   X     sku_ID   brand_ID rep.i..60. X1 X2        X3        X4        X5
## 1 1 067b673f2b 9b0d3a5fc6          1  4  6  51.40385  42.65632  4.942529
## 2 2 068f4481b3 99d41501ff          1  4  2 205.52400 204.95916 81.880040
## 3 3 1a2362c248 9b0d3a5fc6          1  4  7  73.43883  63.41571  6.899135
## 4 4 1d3b8e84d1 4efb032b5a          1  4  2 124.58333 118.60714 39.000000
## 5 5 2523d051fd 99d41501ff          1  4  2 199.25362 200.00000  8.503937
## 6 6 2ddb64e05a 0b0f75e8d5          1  4  5  52.16970  53.24000  5.636364
##          X6        X7 X8        X9        X10
## 1  0.000000 8.8276820  0  42.65632 43.187261
## 2  8.530746 2.4486055  0 204.95916  0.000000
## 3  0.000000 9.1739673  0  63.41571 59.793036
## 4  0.000000 9.3928571  0 118.60714  0.000000
## 5 22.929134 2.2598425  0 200.00000 64.307087
## 6  0.000000 0.6969697  0  53.24000  4.668485
```

```
head(train_vcui)
```

```
##   date    user_ID user_level plus gender age marital_status education
## 1    1 004d88db13          4    1      1  30              1         3
## 2    1 006bc1d136          4    1     -1  30              1         4
## 3    1 00e0fcf701          4    1     -1  30              1         4
## 4    1 00fbda605c          4    0     -1  30              0         3
## 5    1 016c1ee449          4    0     -1  30              1         4
## 6    1 02033e9b30          4    1      1  30              1         3
##   city_level purchase_power X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 X14
## 1          1              2  0  0  0  0  0  0  0  0  0   0   0   0   0   0
## 2          1              2  0  1  0  0  0  0  0  0  0   0   0   0   0   0
## 3          2              2  0  1  0  0  0  0  0  0  0   0   0   0   0   0
## 4          1              2  0  1  0  0  0  0  0  0  0   0   0   0   0   0
## 5          2              3  0  0  0  0  0  0  0  0  0   0   0   0   0   0
## 6          1              2  0  0  0  0  0  0  0  0  0   0   0   0   0   0
##   X15 X16 X17 X18 X19 X20 X21 X22 X23 X24 X25 X26 X27 X28 X29 X30 X31 X32
## 1   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
## 3   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
## 5   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 6   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

```
##   X33 X34 X35 X36 X37 X38 X39 X40 X41 X42 X43 X44 X45 X46 X47 X48 X49 X50
## 1   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
## 3   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
## 5   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 6   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##   X51 X52 X53 X54 X55 X56 X57 X58 X59 X60
## 1   0   0   0   0   0   0   0   0   0   0
## 2   0   6   0   0   0   0   0   0   0   1
## 3   0   6   0   0   0   0   0   0   0   1
## 4   0   6   0   0   0   0   0   0   0   1
## 5   0   0   0   0   0   0   0   0   0   0
## 6   0   0   0   0   0   0   0   0   0   0
```

```
print(colnames(train_vcui))
```

```
##  [1] "date"           "user_ID"        "user_level"     "plus"
##  [5] "gender"         "age"            "marital_status" "education"
##  [9] "city_level"     "purchase_power" "X1"             "X2"
## [13] "X3"             "X4"             "X5"             "X6"
## [17] "X7"             "X8"             "X9"             "X10"
## [21] "X11"            "X12"            "X13"            "X14"
## [25] "X15"            "X16"            "X17"            "X18"
## [29] "X19"            "X20"            "X21"            "X22"
## [33] "X23"            "X24"            "X25"            "X26"
## [37] "X27"            "X28"            "X29"            "X30"
## [41] "X31"            "X32"            "X33"            "X34"
## [45] "X35"            "X36"            "X37"            "X38"
## [49] "X39"            "X40"            "X41"            "X42"
## [53] "X43"            "X44"            "X45"            "X46"
## [57] "X47"            "X48"            "X49"            "X50"
## [61] "X51"            "X52"            "X53"            "X54"
## [65] "X55"            "X56"            "X57"            "X58"
## [69] "X59"            "X60"
```

```
print(colnames(train_skui))
```

```
##  [1] "X"          "sku_ID"     "brand_ID"   "rep.i..60." "X1"
##  [6] "X2"         "X3"         "X4"         "X5"         "X6"
## [11] "X7"         "X8"         "X9"         "X10"
```

```
dim(train_skui)
```

```
## [1] 1200   14
```

```
dim(train_vcui)
```

```
## [1] 9020   70
```

```
train_vcui_mod = train_vcui
# train_vcui_mod$user_ID = as.numeric(seq(1,length(train_vcui$user_ID)))
# train_vcui_mod$user_ID = seq(1,length(train_vcui$user_ID))
train_vcui_mod$user_ID = as.numeric(train_vcui$user_ID)
train_vcui_mod = train_vcui_mod[2:70]
head(train_vcui_mod)
```

```
##   user_ID user_level plus gender age marital_status education city_level
```

```
## 1       1          4    1      1  30              1         3          1
## 2       2          4    1     -1  30              1         4          1
## 3       3          4    1     -1  30              1         4          2
## 4       4          4    0     -1  30              0         3          1
## 5       5          4    0     -1  30              1         4          2
## 6       6          4    1      1  30              1         3          1
##   purchase_power X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 X14 X15 X16
## 1              2  0  0  0  0  0  0  0  0  0   0   0   0   0   0   0   0
## 2              2  0  1  0  0  0  0  0  0  0   0   0   0   0   0   0   0
## 3              2  0  1  0  0  0  0  0  0  0   0   0   0   0   0   0   0
## 4              2  0  1  0  0  0  0  0  0  0   0   0   0   0   0   0   0
## 5              3  0  0  0  0  0  0  0  0  0   0   0   0   0   0   0   0
## 6              2  0  0  0  0  0  0  0  0  0   0   0   0   0   0   0   0
##   X17 X18 X19 X20 X21 X22 X23 X24 X25 X26 X27 X28 X29 X30 X31 X32 X33 X34
## 1   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
## 3   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
## 5   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 6   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##   X35 X36 X37 X38 X39 X40 X41 X42 X43 X44 X45 X46 X47 X48 X49 X50 X51 X52
## 1   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   6
## 3   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   6
## 4   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   6
## 5   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 6   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##   X53 X54 X55 X56 X57 X58 X59 X60
## 1   0   0   0   0   0   0   0   0
## 2   0   0   0   0   0   0   0   1
## 3   0   0   0   0   0   0   0   1
## 4   0   0   0   0   0   0   0   1
## 5   0   0   0   0   0   0   0   0
## 6   0   0   0   0   0   0   0   0
```

```
test_vcui_mod = test_vcui
# test_vcui_mod$user_ID = as.numeric((seq(1,length(test_vcui$user_ID))))
test_vcui_mod$user_ID = as.numeric(test_vcui$user_ID)
# levels(droplevels(test_vcui_mod$user_ID))
test_vcui_mod = test_vcui_mod[2:70]
head(test_vcui_mod)
```

```
##   user_ID user_level plus gender age marital_status education city_level
## 1       1          4    1      1  30              1         3          1
## 2       2          4    1     -1  30              1         4          1
## 3       3          4    1     -1  30              1         4          2
## 4       4          4    0     -1  30              0         3          1
## 5       5          4    0     -1  30              1         4          2
## 6       6          4    1      1  30              1         3          1
##   purchase_power X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 X14 X15 X16
## 1              2  0  0  0  0  0  0  0  0  0   0   0   0   0   0   0   0
## 2              2  0  0  0  0  0  0  0  0  0   0   0   0   0   0   0   0
## 3              2  0  0  0  0  0  0  0  0  0   0   0   0   0   0   0   0
## 4              2  0  0  0  0  0  0  0  0  0   0   0   0   0   0   0   0
## 5              3  0  0  0  0  0  0  0  0  0   0   0   0   0   0   0   0
```

```
## 6              2  0  0  0  0  0  0  0  0  0   0   0   0   0   0   0   0
##   X17 X18 X19 X20 X21 X22 X23 X24 X25 X26 X27 X28 X29 X30 X31 X32 X33 X34
## 1   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
## 3   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
## 5   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 6   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##   X35 X36 X37 X38 X39 X40 X41 X42 X43 X44 X45 X46 X47 X48 X49 X50 X51 X52
## 1   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
## 3   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
## 5   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 6   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##   X53 X54 X55 X56 X57 X58 X59 X60
## 1   0   0   0   0   0   0   0   0
## 2   0   0   0   0   0   0   0   0
## 3   0   0   0   0   0   0   0   0
## 4   0   0   0   0   0   0   0   0
## 5   0   0   0   0   0   0   0   0
## 6   0   0   0   0   0   0   0   0
```

```
dim(train_vcui_mod)
```

```
## [1] 9020   69
```

```
dim(test_vcui_mod)
```

```
## [1] 4510   69
```

```
# library(tidyr)
# cuinformation <- read.csv('train_vcuinformation.csv')
# delete <- c()
# for(i in 1:dim(cuinformation)[1]){
#   if(sum(cuinformation[i, 11:70]) == 0){
#     delete <- c(delete, i)
#   }
# }
# cuinformation <- cuinformation[-delete, ]
```

USER CHOICE PREDICTION MODEL

```
# model = lm(user_ID ~ ., data = train_vcui)
# summary(model)
model = lm(user_ID ~ ., data = train_vcui_mod)
summary(model)
```

```
##
## Call:
## lm(formula = user_ID ~ ., data = train_vcui_mod)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -255.698 -109.219    1.159  108.019  242.401
```

```
##
## Coefficients: (9 not defined because of singularities)
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)     2.270e+02  1.223e+01  18.557  < 2e-16 ***
## user_level     -1.374e+01  1.902e+00  -7.227 5.32e-13 ***
## plus            4.084e+00  3.031e+00   1.347  0.17789
## gender         -8.718e+00  1.698e+00  -5.134 2.90e-07 ***
## age            -5.606e-01  1.729e-01  -3.242  0.00119 **
## marital_status  2.541e+01  3.263e+00   7.785 7.72e-15 ***
## education       1.127e+01  2.832e+00   3.979 6.98e-05 ***
## city_level      3.710e+00  1.301e+00   2.853  0.00435 **
## purchase_power -2.244e+00  2.932e+00  -0.765  0.44416
## X1              1.709e+01  2.393e+01   0.714  0.47519
## X2              2.102e+00  8.168e+00   0.257  0.79689
## X3              5.906e+00  1.976e+01   0.299  0.76503
## X4                     NA         NA      NA       NA
## X5              1.816e+01  1.724e+01   1.053  0.29214
## X6                     NA         NA      NA       NA
## X7              2.007e+02  2.206e+02   0.910  0.36310
## X8             -3.195e+00  5.406e+00  -0.591  0.55460
## X9                     NA         NA      NA       NA
## X10             1.506e+01  2.496e+01   0.603  0.54634
## X11            -1.241e+01  8.795e+01  -0.141  0.88781
## X12            -5.599e+00  2.665e+01  -0.210  0.83362
## X13            -6.288e+01  3.932e+01  -1.599  0.10986
## X14            -5.505e+01  4.634e+01  -1.188  0.23493
## X15                    NA         NA      NA       NA
## X16            -1.019e+02  8.765e+01  -1.163  0.24487
## X17             5.363e+00  6.733e+01   0.080  0.93652
## X18             2.727e+00  4.328e+00   0.630  0.52859
## X19             4.561e+00  1.349e+01   0.338  0.73535
## X20             4.547e+01  9.131e+01   0.498  0.61850
## X21             1.952e+02  1.302e+02   1.499  0.13402
## X22            -3.195e+01  2.239e+01  -1.427  0.15367
## X23            -7.226e+01  7.607e+01  -0.950  0.34220
## X24             2.936e+01  2.161e+01   1.359  0.17413
## X25            -1.268e+00  8.680e+00  -0.146  0.88385
## X26             3.780e+01  1.614e+01   2.342  0.01922 *
## X27                    NA         NA      NA       NA
## X28            -8.497e+00  1.028e+02  -0.083  0.93411
## X29            -8.298e+00  1.050e+01  -0.790  0.42951
## X30             6.461e-13  1.824e+02   0.000  1.00000
## X31            -1.481e+02  1.291e+02  -1.147  0.25153
## X32            -4.740e+01  1.290e+02  -0.367  0.71331
## X33             6.103e+00  1.078e+01   0.566  0.57123
## X34             6.128e+00  3.487e+00   1.757  0.07891 .
## X35                    NA         NA      NA       NA
## X36            -2.547e+01  3.711e+01  -0.686  0.49259
## X37            -7.947e+00  6.932e+00  -1.146  0.25166
## X38            -1.141e+02  7.508e+01  -1.520  0.12851
## X39            -5.391e+01  6.768e+01  -0.797  0.42574
## X40                    NA         NA      NA       NA
## X41             1.281e+02  5.215e+01   2.456  0.01407 *
## X42             1.035e+01  1.773e+01   0.584  0.55924
```

```
## X43            -1.274e+02  9.273e+01  -1.374  0.16940
## X44                    NA         NA      NA       NA
## X45            -2.898e+01  1.799e+02  -0.161  0.87198
## X46             1.852e+01  8.401e+01   0.221  0.82548
## X47             1.891e+01  2.468e+01   0.766  0.44355
## X48             1.098e+01  1.997e+01   0.550  0.58241
## X49            -1.145e+01  1.389e+02  -0.082  0.93432
## X50             1.096e+00  2.964e+00   0.370  0.71170
## X51             1.238e+01  1.025e+02   0.121  0.90388
## X52            -3.355e+00  2.940e+00  -1.141  0.25382
## X53            -1.617e+01  1.174e+01  -1.377  0.16841
## X54            -1.352e+01  1.689e+01  -0.800  0.42348
## X55            -2.297e+01  1.620e+01  -1.418  0.15624
## X56            -1.886e+01  9.500e+00  -1.985  0.04717 *
## X57             1.702e+01  9.137e+01   0.186  0.85227
## X58            -1.652e+02  9.127e+01  -1.810  0.07036 .
## X59                    NA         NA      NA       NA
## X60             1.068e+01  1.217e+01   0.878  0.38015
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 129 on 8960 degrees of freedom
## Multiple R-squared:  0.02519,    Adjusted R-squared:  0.01877
## F-statistic: 3.924 on 59 and 8960 DF,  p-value: < 2.2e-16
```

```
# factor(x = character(), levels, labels = levels, ordered = is.ordered(x))
# model = lm(factor(user_ID) ~ X1, data = train_vcui)
# res = model.matrix(~rank, data = train_vcui)
print("R^2 0.025")
```

```
## [1] "R^2 0.025"
```

```
pred = predict(model, newdata=test_vcui_mod)
```

```
## Warning in predict.lm(model, newdata = test_vcui_mod): prediction from a
## rank-deficient fit may be misleading
```

```
SSE = sum((test_vcui_mod$user_ID - pred)^2)
SST = sum((test_vcui_mod$user_ID - mean(train_vcui_mod$user_ID))^2)
OSR2 = 1 - SSE/SST
OSR2
```

```
## [1] -0.01170546
```

PERIPHERAL MODELS

Purchase Power Model

```
model_pp = lm(purchase_power ~ date + user_level + plus + gender + age + marital_status + education + c
summary(model_pp)
```

```
##
## Call:
## lm(formula = purchase_power ~ date + user_level + plus + gender +
##     age + marital_status + education + city_level, data = train_vcui)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.32798 -0.23765 -0.15044 -0.04175  2.71724
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)     2.397e+00  3.698e-02  64.827  < 2e-16 ***
## date            1.929e-17  8.492e-04   0.000 1.000000
## user_level     -2.146e-02  6.830e-03  -3.142 0.001684 **
## plus           -1.216e-01  1.080e-02 -11.261  < 2e-16 ***
## gender          1.545e-03  6.095e-03   0.254 0.799887
## age             2.370e-03  6.200e-04   3.823 0.000133 ***
## marital_status  4.517e-02  1.173e-02   3.853 0.000118 ***
## education      -7.427e-02  1.012e-02  -7.337 2.37e-13 ***
## city_level      2.909e-02  4.652e-03   6.255 4.17e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4651 on 9011 degrees of freedom
## Multiple R-squared:  0.03938,    Adjusted R-squared:  0.03853
## F-statistic: 46.17 on 8 and 9011 DF,  p-value: < 2.2e-16
```

```
print("R^2 of 0.039")
```

```
## [1] "R^2 of 0.039"
```

```
# vif(model_pp)
# Anova(model_pp)

model2_pp = lm(purchase_power ~ user_level + plus + age + marital_status + education + city_level, data
summary(model2_pp)
```

```
##
## Call:
## lm(formula = purchase_power ~ user_level + plus + age + marital_status +
##     education + city_level, data = train_vcui)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.32803 -0.23833 -0.14881 -0.04284  2.71956
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)     2.3940234  0.0337787  70.874  < 2e-16 ***
## user_level     -0.0210697  0.0066550  -3.166 0.001551 **
## plus           -0.1214471  0.0107748 -11.271  < 2e-16 ***
## age             0.0023763  0.0006195   3.836 0.000126 ***
## marital_status  0.0448646  0.0116610   3.847 0.000120 ***
## education      -0.0740681  0.0100897  -7.341  2.3e-13 ***
## city_level      0.0292175  0.0046256   6.317  2.8e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.465 on 9013 degrees of freedom
```

```
## Multiple R-squared:  0.03937,    Adjusted R-squared:  0.03873
## F-statistic: 61.57 on 6 and 9013 DF,  p-value: < 2.2e-16
```

```
print("R^2 of 0.039")
```

```
## [1] "R^2 of 0.039"
```

```
# vif(model2_pp)
# Anova(model2_pp)
```

```
pred = predict(model2_pp, newdata=test_vcui_mod)
SSE = sum((test_vcui_mod$purchase_power - pred)^2)
SST = sum((test_vcui_mod$purchase_power - mean(train_vcui_mod$purchase_power))^2)
OSR2 = 1 - SSE/SST
OSR2
```

```
## [1] 0.03937284
```

Feature Relationships Model

```
# model_X1 = lm(X1 ~ . - date + user_ID + user_level + plus + gender + age + marital_status + education
model_X1 = lm(X1 ~ X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10, data = train_skui)
summary(model_X1)
```

```
##
## Call:
## lm(formula = X1 ~ X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10,
##     data = train_skui)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.7515 -0.4924  0.2164  0.5940  2.0595
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.8176642  0.0876972  32.129  < 2e-16 ***
## X2           0.0611824  0.0122832   4.981 7.26e-07 ***
## X3           0.0049262  0.0033779   1.458   0.1450
## X4           0.0055328  0.0037861   1.461   0.1442
## X5          -0.0079563  0.0013720  -5.799 8.54e-09 ***
## X6          -0.0195362  0.0139060  -1.405   0.1603
## X7           0.0188133  0.0241970   0.778   0.4370
## X8           0.3774928  0.2010935   1.877   0.0607 .
## X9          -0.0028013  0.0047498  -0.590   0.5555
## X10          0.0000534  0.0017467   0.031   0.9756
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.015 on 1190 degrees of freedom
## Multiple R-squared:  0.08462,    Adjusted R-squared:  0.07769
## F-statistic: 12.22 on 9 and 1190 DF,  p-value: < 2.2e-16
```

```
pred = predict(model_X1, newdata=test_skui)
SSE = sum((test_skui$X1 - pred)^2)
SST = sum((test_skui$X1 - mean(train_skui$X1))^2)
```

```
OSR2 = 1 - SSE/SST
OSR2
```

```
## [1] 0.06013254
```

recommdation algorithm

Yifan Li

```
usernumeric<-function()
{
user=read.csv("JD_user_data.csv")
user$first_order_month=as.numeric(user$first_order_month)
library(stringr)
user$gender=str_replace_all(user$gender,pattern="F",'-1')
user$gender=str_replace_all(user$gender,pattern="M",'1')
user$gender=str_replace_all(user$gender,pattern="U",'0')
sum(user$age=='U')/nrow(user)#only 0.1234578 users' age is unkown so we remove this part.
user=subset(user,user$age!='U')
user$age=str_replace_all(user$age,pattern="26-35",'30')
user$age=str_replace_all(user$age,pattern="<=15",'10')
user$age=str_replace_all(user$age,pattern=">=56",'60')
user$age=str_replace_all(user$age,pattern="36-45",'40')
user$age=str_replace_all(user$age,pattern="16-25",'20')
user$age=str_replace_all(user$age,pattern="46-55",'50')
sum("U"==user$marital_status)/nrow(user)#only 0.09436 users' marital_status is unkown
user=subset(user,user$marital_status!='U')
user$marital_status=str_replace_all(user$marital_status,pattern="M",'1')
user$marital_status=str_replace_all(user$marital_status,pattern="S",'0')
sum(user$purchase=='-1')/nrow(user)
sum(user$edu=='-1')/nrow(user)
sum(user$city=='-1')/nrow(user)
sum(user$gender=='0')/nrow(user)
user=subset(user,user$purchase_power!='-1')
user=subset(user,user$education!='-1')
user=subset(user,user$city_level!='-1')
user=subset(user,user$gender!='0')

return(user)
}
```

```
customer=Vcu
normalize <- function(x)
{x=as.numeric(x)
return ((x - mean(x)) / sd(x))
  }
for(i in 1:ncol(customer))
{
  customer[,i]=normalize(customer[,i])
}
cluster=vector(length=30)
for(i in 1:30)
{
```

```
cluster[i]=sum(kmeans(customer,i)$withinss)
}
plot(cluster)
cluster=kmeans((customer),4)
center=cluster$cluster
cluster1=kmeans((customer),6)
temp=cbind(aimuser,Vcu,center)
temp=split(temp,temp$center)
cl1=temp$'1'
cl2=temp$'2'
cl3=temp$"3"
cl4=temp$"4"
#using decision tree to predict the cluster
train=cbind(aimuser,center)[1:350,-1]
test=cbind(aimuser,center)[351:451,-1]
for(i in 1: ncol(train))
{
  train[,i]=as.numeric(train[,i])
}
for(i in 1: ncol(test))
{
  test[,i]=as.numeric(test[,i])
}
train$center=as.factor(train$center)

train.cart = train(center~.-center,
                   data = train,
                   method = "rpart",
                   tuneGrid = data.frame(cp=seq(0, 0.1, 0.001)),
                   trControl = trainControl(method="cv", number=5))

mod.cart <- train.cart$finalModel
test$center=as.factor(test$center)
pred.cart <- predict(mod.cart, newdata =test, type = "class")

table(test$center,pred.cart)
```

subset the user and sku that has been purchase enough time to be analysis, we will only analysis the 1p sku

```
sku=read.csv("../data/JD_sku_data.csv")
order=read.csv('../data/JD_order_data.csv')
aimsku=as.data.frame(table(order$sku_ID))
aimsku=aimsku[order(aimsku$Freq,decreasing=T),]
names(aimsku)[1]='sku_ID'

aimsku=merge(sku,aimsku)
aimsku=subset(aimsku,aimsku$type==1)

aimsku=subset(aimsku,aimsku$Freq>1000)
aimorder=subset(order,order$sku_ID %in%aimsku$sku_ID)
aimuser=as.data.frame(table(aimorder$user_ID))
sum(aimuser$Freq>6)
```

```r
aimuser=subset(aimuser,aimuser$Freq>4)
write.csv(aimuser,"aimuesr.csv")
write.csv(aimsku,"aimsku.csv")
```

```r
#order=read.csv('../data/JD_order_data.csv')
# aimuser=read.csv("aimuesr.csv")[,-1]
# aimsku=read.csv('aimsku.csv')[,-1]
#
# aimorder=subset(order,order$sku_ID %in%aimsku$sku_ID)
# aimorder=subset(order,order$user_ID %in%aimuser$Var1)
#
# aimorder$user_ID=as.character(aimorder$user_ID)
# aimorder$sku_ID=as.character(aimorder$sku_ID)
#
# P=matrix(0,nrow=nrow(aimuser),ncol=nrow(aimsku))
# i=1
# j=1
calculateP<-function(date)
{P=matrix(0,nrow=nrow(aimuser),ncol=nrow(aimsku))
for( i in 1:nrow(aimuser))
{
  for(j in 1:nrow(aimsku))
  P[i,j]=nrow(subset(aimorder,aimorder$user_ID==aimuser$user_ID[i] & aimorder$sku_ID==aimsku$sku_ID[j]&a

}
return(P)
}

sum(as.numeric(calculateP(1)))

trainorder=subset(aimorder,aimorder$order_date<43180)
#trainfre=as.matrix(table(trainorder$sku_ID))
#write.csv(trainfre,'trainfre.csv')
#trainfre=read.csv("trainfre.csv")
#trainfre$X=as.character(trainfre$X)
# names(trainfre)[1]="sku_ID"
# trainorder=merge(aimsku,trainfre,all.x=T)
# trainorder[is.na(trainorder)]=0
```

```r
obj=function(date,Vcu,Vsk,Y)#input: P matrix, a matrix of customer vectors, and a matix of sku vectors,
{temp=0

  tem=log(Y)

  for(i in 1:nrow(X))
  {
    temp=sum(P[i,]%*%t(P[i,]==0)*tem[i,,])+temp
  }


  return(temp)
}
```

```
Vskpromo=function(Vsk,date)
{
  for(i in 1:nrow(Vsk))
  {
    Vsk[i,10]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i] & aimskuorder$order_date==
    Vsk[i,11]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i] &aimskuorder$order_date==
     Vsk[i,12]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i] & aimskuorder$order_date=
    Vsk[i,13]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i]& aimskuorder$order_date==
   Vsk[i,14]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i]& aimskuorder$order_date==(
   Vsk[i,15]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i]& aimskuorder$order_date==(




  }
  Vsk[is.na(Vsk)]=0
  return(Vsk)
}
```

constructing the vectors matrix, initialize

```
#order=read.csv('../data/JD_order_data.csv')
aimskuorder=subset(order,order$sku_ID %in% aimsku$sku_ID)
hy=6#hyperparameter to decide the dim of the vsctors, must >=4 for 4 kind of promotion
#users=usernumeric()
#aimuser=read.csv("aimuesr.csv")
aimuser=subset(users,users$user_ID %in% aimuser$Var1)
for(i in 1:ncol(aimuser))
{
aimuser[,i]=as.numeric(aimuser[,i])
}
temp=matrix(0,nrow=nrow(aimuser),ncol=hy)

Vcu=as.matrix(cbind(aimuser,temp))
temp=matrix(0,nrow=nrow(aimsku),ncol=ncol(Vcu))
Vsk=temp

aimorder$final_unit_price=as.numeric(aimorder$final_unit_price)
aimorder$direct_discount_per_unit=as.numeric(aimorder$direct_discount_per_unit)
aimorder$quantity_discount_per_unit=as.numeric(aimorder$quantity_discount_per_unit)
aimorder$bundle_discount_per_unit=as.numeric(aimorder$bundle_discount_per_unit)
aimorder$coupon_discount_per_unit=as.numeric(aimorder$coupon_discount_per_unit)
aimorder$gift_item=as.numeric(aimorder$gift_item)
aimskuorder$sku_ID=as.character(aimskuorder$sku_ID)
aimuser=read.csv("aimuesr.csv")

popularity=rowsum(aimskuorder$quantity,aimskuorder$sku_ID)
write.csv(popularity,"popularity.csv")
```

```
Vcugradient<-function(P,Vcu,Vsk,Y,Vsku)
{
  #VCU gradient

  temp=matrix(0,nrow=nrow(Vcu),ncol=ncol(Vcu))
```

```
  #gradient for Vcu

  for( i in 1:nrow(Vcu))
  {
for(j1 in 1:nrow(Vsk))
   {for(j2 in 1:nrow(Vsk))

    {
      temp[i,]=P[i,j1]*(P[i,j2]==0)*(Vsku[j1,j2,])*min(100000,max(-100000,exp(as.numeric(Vcu[i,]%*%Vsku

    }
}
  }

return(temp)


}
Vskgradient<-function(P,Vcu,Vsk,Y,Vsku)
{ temp=matrix(0,nrow=nrow(Vsk),ncol=ncol(Vcu))
for( j in 1:nrow(Vsk)){
  for( k in 1:nrow(Vsk))
  {
    for(i in 1:nrow(Vcu))
    {
      temp[j,]=(P[i,k]*(P[i,j]==0)*(-Vcu[i,])*min(100000,max(-100000,as.numeric(exp(Vcu[i,]%*%Vsku[j,k,]
    }
  }
}
return(temp)
}

Vcugradientfast<-function(P,Vcu,Vsk,Y,Vsku)
{
  #VCU gradient

  temp=matrix(0,nrow=nrow(Vcu),ncol=ncol(Vcu))
  #gradient for Vcu

  for( i in 1:nrow(Vcu))
  {
for(j1 in which(P[i,]!=0))
   {for(j2 in which(P[i,]==0))

    {
      temp[i,]=(Vsku[j1,j2,])*min(100000,max(-100000,exp(as.numeric(Vcu[i,]%*%Vsku[j2,j1,]))*Y[i,j1,j2]]

    }
}
  }

return(temp)
```

```
}
Vskgradientfast<-function(P,Vcu,Vsk,Y,Vsku)
{ temp=matrix(0,nrow=nrow(Vsk),ncol=ncol(Vcu))
for( j in 1:nrow(Vsk)){
  for( k in which(P[i,]!=0))
  {
    for(i in 1:nrow(Vcu))
    {
      temp[j,]=(P[i,k]*(P[i,j]==0)*(-Vcu[i,])*min(100000,max(-100000,as.numeric(exp(Vcu[i,]%*%Vsku[j,k,]
    }
  }
}
return(temp)
}
```

```
#VCU

it=10
objec1=numeric(it)
tempob=ob=-8000000
ste=0.000001
for(ite in 1:it)
{objec1[ite]=tempob
  if( tempob<ob)
  {
    ste=ste/10
  }else
    ob=tempob
  tempob=0
for( date in 1:20)
{
P=pmatrix[date,,]
Vsk[,9:14]=vskyuan[date,,9:14]
for( jj in 1:1)

{

  X=Vcu%*%t(Vsk)

  Y=array(0,dim=c(nrow(X),ncol(X),ncol(X)))
  for(i in 1:nrow(X))
  {
    for(j in 1:ncol(X))
    {
      for( k in 1:ncol(X))
      {
      Y[i,j,k]=1/(1+exp(-X[i,j]+X[i,k]))
      }

    }

  }
  Y[Y==0]=0.00000001
```

```
tempob=obj(date,Vcu,Vsk,Y)+tempob

  Vsku=array(0,dim=c(nrow(Vsk),nrow(Vsk),ncol(Vsk)))
  for( i in 1:nrow(Vsk))
   {for( j in 1:nrow(Vsk))
    {Vsku[i,j,]=Vsk[i,]-Vsk[j,]}}
vcug=Vcugradientfast(P,Vcu,Vsk,Y,Vsku)
vskg=Vskgradient(P,Vcu,Vsk,Y,Vsku)
Vcu[,9:ncol(Vcu)]=Vcu[,9:ncol(Vcu)]+vcug[,9:ncol(Vcu)]*ste
Vsk[,1:8]=Vsk[,1:8]+vskg[,1:8]*ste


}
}
}
data=as.data.frame(cbind(c(1:nrow(rbind(as.matrix(objv),as.matrix(objv1)))),rbind(as.matrix(objv),as.ma
library(ggplot2)
ggplot(data=data,aes(x=V1,y=V2))+geom_point()+labs(x='iteration',y='obj',title="training process")
```

```
write.csv(Vcu,"result/vcu_4_2.csv")
write.csv(Vsk,"result/vsk_1000_vector_2.csv")
nrow(Vcu)
nrow(aimuser)
nrow(aimsku)
write.csv(aimsku,"result/1000_sku_2.csv")
write.csv(aimuser,"result/4_user_2.csv")
```

calculate bechmark

```
bench=vector(length=20)
for(i in 1:20)
{P=calculateP(i)
Vsk=Vskpromo(Vsk,i)
  bench[i]=obj(date,Vcu,Vsk,Y)
}

test0=testbench=vector(length=10)
for( t in 21:30)
{P=calculateP(t)
# Vsk=Vskall(Vsk,t)
# X=Vcu%*%t(Vsk)
#
#   Y=array(0,dim=c(nrow(X),ncol(X),ncol(X)))
#   for(i in 1:nrow(X))
#   {
#     for(j in 1:ncol(X))
#     {
#       for( k in 1:ncol(X))
#       {
#       Y[i,j,k]=1/(1+exp(-X[i,j]+X[i,k]))
#       }
#
#     }
```

```
#
#   }
#   Y[Y==0]=0.00000001
#   test0[t-20]=obj(date,Vcu,Vsk,Y)
  temp=0
  for(i in 1:nrow(X))
  {
    temp=sum(P[i,]%*%t(P[i,]==0)*log(1/2))+temp
  }

  testbench[t-20]=temp
}

test0=as.matrix(test0)
testbench=as.matrix(testbench)
data=as.data.frame(cbind(c(1:10),test,testbench))
library(ggplot2)
ggplot(data=data,aes(x=V1))+geom_point(aes(y=test0-test))+labs(x='date',y='difference',title="objective
```

```
customer=Vcu[,10:15]
```

this part we only consider the sku's information(adding historical price to it)

```
Vskall=function(Vsk,date)
{
  if(date==1)
  {
    for(i in 1:nrow(Vsk))
  {  Vsk[i,2]=as.numeric(subset(aimsku, aimsku$sku_ID == aimsku$sku_ID[i])$attribute2)
    Vsk[i,1]=as.numeric(subset(aimsku, aimsku$sku_ID == aimsku$sku_ID[i])$attribute1)
    Vsk[i,3]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i] & aimskuorder$order_date==
    Vsk[i,4]=0
    Vsk[i,5]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i] & aimskuorder$order_date==
    Vsk[i,6]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i] &aimskuorder$order_date==(
     Vsk[i,7]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i] & aimskuorder$order_date=
    Vsk[i,8]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i]& aimskuorder$order_date==(
   Vsk[i,9]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i]& aimskuorder$order_date==(da
   Vsk[i,10]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i]& aimskuorder$order_date==(
  }
  }
    else
    {
  for(i in 1:nrow(Vsk))
  {  Vsk[i,2]=as.numeric(subset(aimsku, aimsku$sku_ID == aimsku$sku_ID[i])$attribute2)
    Vsk[i,1]=as.numeric(subset(aimsku, aimsku$sku_ID == aimsku$sku_ID[i])$attribute1)
    Vsk[i,3]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i] & aimskuorder$order_date==
    Vsk[i,4]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i] & aimskuorder$order_date==
    Vsk[i,5]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i] & aimskuorder$order_date==
    Vsk[i,6]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i] &aimskuorder$order_date==(
     Vsk[i,7]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i] & aimskuorder$order_date=
    Vsk[i,8]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i]& aimskuorder$order_date==(
   Vsk[i,9]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i]& aimskuorder$order_date==(da
   Vsk[i,10]=mean(subset(aimskuorder, aimskuorder$sku_ID == aimsku$sku_ID[i]& aimskuorder$order_date==(
```

```
  }

  }
  Vsk[is.na(Vsk)]=0
  return(Vsk)
}
Vska=Vskall(Vsk,date)
```

```
#initialize
# Vsk=matrix(0,nrow=60,ncol=10)


#Vcu=matrix(0,nrow=451,ncol=11)

#Vcu=as.matrix(read.csv('Vcu_all_without_the_first_day_order.csv')[,-1])
it=2
objec=numeric(it)
tempob=ob=-763.9843

ste=0.01
for(ite in 1:it)
{objec[ite]=tempob
  if( tempob<ob)
  {
    ste=ste/10
  }else
    ob=tempob
  tempob=0
  #temp=sample(c(1:19),5)
  temp=c(1:19)
for( day in temp)
{date=day+1
P=pmatrix[date,,]
Vsk=vskmatrixnormalize[date,,]
for( t in 1:1)

{

  X=Vcu%*%t(Vsk)

  Y=array(0,dim=c(nrow(X),ncol(X),ncol(X)))
  for(i in 1:nrow(X))
  {
    for(j in 1:ncol(X))
    {
      for( k in 1:ncol(X))
      {
      Y[i,j,k]=1/(1+exp(-X[i,j]+X[i,k]))
      }

    }

  }
```

```
  Y[Y==0]=0.00000001
  objv3=obj(date,Vcu,Vsk,Y)/sum(P)
  tempob=objv3+tempob
  Vsku=array(0,dim=c(nrow(Vsk),nrow(Vsk),ncol(Vsk)))
  for( i in 1:nrow(Vsk))
   {for( j in 1:nrow(Vsk))
    {Vsku[i,j,]=Vsk[i,]-Vsk[j,]}}
vcug=Vcugradient(P,Vcu,Vsk,Y,Vsku)
Vcu=Vcu+vcug*ste



}
}
}
sd(Vcu[,1])
summary=summary(Vcu)
as.numeric(summary[4,])/(as.numeric(summary[6,])-as.numeric(summary[1,]))
```

test the rank accuracy

```
# vskyuan=array(0,dim=c(30,nrow(Vsk),ncol(Vsk)))
# for( i in 1:30)
# {vskyuan[i,,]=Vskpromo(Vsk,i)
# }

 # pmatrix=array(0,dim=c(30,nrow(P),ncol(P)))
# vskmatrix=array(0,dim=c(30,nrow(Vsk),ncol(Vsk)))
# for(i in 1:30)
#  {
# pmatrix[i,,]=calculateP(i)
# }
#
# for(i in 1:20)
# {
# vskmatrix[i,,]=Vskall(Vsk,i)
# }
# benchmarkresult=order(trainorder$V1,decreasing=T)
# benchmarkresult=rep(1, times = 451) %*% t(benchmarkresult)
result=matrix(0,nrow=nrow(aimuser),ncol=55)

acc=0
error=0
acc1=0
error1=0
accuracy=numeric(55)
accuracy1=numeric(55)
# p=matrix(0,nrow=451,ncol=60)
# for( i in 1:30)
# {
#   p=p+pmatrix[i,,]
# }
for(k in 1:55)
```

```
{acc=0
error=0
acc1=0
error1=0
for(date in 1:10){
  P=pmatrix[date+20,,]
Vsk=vskmatrixnormalize[date+20,,]
X=Vcu%*%t(Vsk)
for( i in 1:nrow(X))
{
  X[i,]=X[i,]-min(X[i,])

}
#X=X*(!p)
for(i in 1:nrow(X))
{
  X[i,]=order(X[i,],decreasing= T)
}
for(i in 1: nrow(Vcu))# the k largest sku
{
  for(h in 1:k)
  {
    result[i,h]=which(X[i,]==h)
  }
}
for(i in 1:nrow(Vcu))
{
  if(Vcu[i,1]!=0&&sum(P[i,-1])!=0)#only test the trained Vcu, and the customer that has purchased that
  {
   # if(sum(which(P[i,]!=0) %in% result[i,])!=0) acc=acc+1 else error=error+1
   # if(sum(which(P[i,]!=0) %in% benchmarkresult[i,1:k])!=0) acc1=acc1+1 else error1=error1+1
    acc=sum(P[i,][result[i,1:k]])+acc
   error=sum(P[i,])+error
   acc1=sum(P[i,][benchmarkresult[i,1:k]])+acc1
  }

}

}
accuracy[k]=acc/error
accuracy1[k]=acc1/error
# accuracy[k]=acc/(acc+error)
# accuracy1[k]=acc1/(acc1+error1)
}

benchmark=numeric(40)
for( i in 1:55)
  benchmark[i]=i/60
data=as.data.frame(cbind(c(1:55),benchmark,accuracy,accuracy1))
library(ggplot2)
ggplot(data=data,aes(x=V1))+geom_point(aes(y=benchmark))+geom_point(aes(y=accuracy,colour="test accurac
```

deal with vsk missing value

```
# mean(vskmatrix[vskmatrix[,j,4]!=0,j,4])#using the average value of the
# temp=vector()
# for(j in 1:60)
#  for(date in 1:28)
#  {
# temp[j]=temp[j]+vskmatrix[date,j,9]==0&&vskmatrix[date+1,j,9]==0&&vskmatrix[date+2,j,9]# caluculate t
# }
#for(j in 1:60)
 #temp=vector()
#vskmatrix[vskmatrix[,j,4]==0,j,4]=NA
# }
# for(j in 1:60)
# {
#   for(date in which(is.na(vskmatrix[,j,3])))
#   {
#
#     if(date==1)
#     {
#       vskmatrix[date,j,3]=vskmatrix[date+2,j,3]
#     }
#
#     if(date==30)
#     {
#       vskmatrix[date,j,3]=vskmatrix[date-2,j,3]
#     }
#
#     if(date<30&&date>1)
#     vskmatrix[date,j,3]=mean(c(vskmatrix[date+1,j,3],vskmatrix[date-1,j,3],vskmatrix[date-2,j,3],vskm
#   }
#
# }
# vskmatrix[is.nan(vskmatrix)]=NA
# for(j in 1:60)
# {
#   for(date in which(is.na(vskmatrix[,j,9])))
#   {
#
#
#     vskmatrix[date,j,]=mean(vskmatrix[,j,9],na.rm=T)
#   }
#
# }


#adding populrity information
# popularity=read.csv("popularity.csv")
# names(popularity)[1]="sku_ID"
# popularity=merge(aimsku,popularity)
# popularity=popularity[,9]
# vskmatrix1=array(0,dim=c(30,60,11))
# for(i in 1:30)
# {
#   vskmatrix1[i,,]=cbind(vskmatrix[i,,],popularity)
```

```
# }

#normalize
vskmatrixnormalize=vskmatrix1
for(i in 1:30)
  {for(j in 1:11)
  {
    vskmatrixnormalize[i,,j]=normalize(vskmatrixnormalize[i,,j])
  }
}
vskmatrixnormalize[is.na(vskmatrixnormalize)]=0

sum(is.nan(vskmatrix))
```

```
result=matrix(0,nrow=nrow(aimuser),ncol=60)

acc=0
error=0
acc1=0
error1=0
accuracy=numeric(60)
accuracy1=numeric(60)
# p=matrix(0,nrow=451,ncol=60)
# for( i in 1:30)
# {
#   p=p+pmatrix[i,,]
# }
for(k in 1:55)
{acc=0
error=0
acc1=0
error1=0
for(date in 1:10){
  P=pmatrix[date,,]
Vsk=vskmatrix1[date,,]
X=Vcu%*%t(Vsk)
for( i in 1:nrow(X))
{
  X[i,]=X[i,]-min(X[i,])

}
 #X=X*(!p)
for(i in 1:nrow(X))
{
  X[i,]=order(X[i,],decreasing=T)
}
for(i in 1: nrow(Vcu))# the k largest sku
{
  for(h in 1:k)
  {
    result[i,h]=which(X[i,]==h+5)
  }
}
```

```
for(i in 1:nrow(Vcu))
{
  if(Vcu[i,1]!=0&&sum(P[i,-1])!=0)#only test the trained Vcu, and the customer that has purchased that
  {
   # if(sum(which(P[i,]!=0) %in% result[i,])!=0) acc=acc+1 else error=error+1
   # if(sum(which(P[i,]!=0) %in% benchmarkresult[i,1:k])!=0) acc1=acc1+1 else error1=error1+1
    acc=sum(P[i,][result[i,1:k]])+acc
   error=sum(P[i,])+error
   acc1=sum(P[i,][benchmarkresult[i,1:k]])+acc1
  }

}

}
accuracy[k]=acc/error
accuracy1[k]=acc1/error
# accuracy[k]=acc/(acc+error)
# accuracy1[k]=acc1/(acc1+error1)
}

benchmark=numeric(40)
for( i in 1:60)
  benchmark[i]=i/60
data=as.data.frame(cbind(c(1:60),benchmark,accuracy,accuracy1))
library(ggplot2)
ggplot(data=data,aes(x=V1))+geom_point(aes(y=benchmark))+geom_point(aes(y=accuracy,colour="red"))+labs(
```

{r eval=False}

```
skuinformation=read.csv('train_skuinformation_missingvalue_removed.csv')
cuinformation=read.csv('train_vcuinformation.csv')
skutest <- read.csv('test_skuinformation_missingvalue_removed.csv')
vcutest <- read.csv('test_vcuinformation.csv')
names(cuinformation)[11:70]=as.character(skuinformation[1:60,1])
names(vcutest)[11:70]=as.character(skutest[1:60,1])
library(tidyr)
temp=pivot_longer(cuinformation,c(11:70))
temp1 <- pivot_longer(vcutest,c(11:70))
skuinformation=as.data.frame(skuinformation)
skutest <- as.data.frame(skutest)
temp=as.data.frame(temp)
temp1 <- as.data.frame(temp1)
names(temp)[11]="sku_ID"
names(temp1)[11]="sku_ID"
train=merge(temp,skuinformation, by = c("date", "sku_ID"))
test=merge(temp1,skutest, by = c("date", "sku_ID"))
train1 <- train
library(car)
```

```
## Loading required package: carData
```

```
logistic <- glm(value~.-date-user_ID-sku_ID-brand_ID, data = train)
vif(logistic)
```

```
##     user_level           plus         gender            age marital_status
##       1.611299       1.226179       1.135888       1.216860       1.291449
##      education     city_level purchase_power             X1             X2
##       1.369711       1.032982       1.040994       1.092437       1.153917
##             X3             X4             X5             X6             X7
##      31.936914      39.831991       1.571821       1.267967       1.128033
##             X8             X9            X10
##       1.142353      63.157328       1.216328
```

```
logistic1 <- glm(value~.-date-user_ID-sku_ID-brand_ID-X9-X4-X3, data = train)
vif(logistic1)
```

```
##     user_level           plus         gender            age marital_status
##       1.611299       1.226179       1.135888       1.216860       1.291449
##      education     city_level purchase_power             X1             X2
##       1.369711       1.032982       1.040994       1.021031       1.071152
##             X5             X6             X7             X8            X10
##       1.141170       1.046736       1.119958       1.048919       1.174647
```

```
summary(logistic1)
```

```
##
## Call:
## glm(formula = value ~ . - date - user_ID - sku_ID - brand_ID -
##     X9 - X4 - X3, data = train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -0.107  -0.006  -0.003   0.001  38.908
##
```

```
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -6.184e-03  2.205e-03  -2.805 0.005039 **
## user_level     -2.274e-04  3.166e-04  -0.718 0.472654
## plus            3.352e-04  5.040e-04   0.665 0.505981
## gender         -2.765e-04  2.824e-04  -0.979 0.327531
## age            -2.043e-05  2.875e-05  -0.710 0.477427
## marital_status  5.039e-04  5.437e-04   0.927 0.354042
## education      -1.628e-04  4.705e-04  -0.346 0.729389
## city_level     -2.949e-04  2.160e-04  -1.365 0.172152
## purchase_power -1.369e-03  4.881e-04  -2.805 0.005024 **
## X1              3.548e-03  2.171e-04  16.343  < 2e-16 ***
## X2             -3.091e-04  9.261e-05  -3.338 0.000844 ***
## X5              4.240e-05  9.182e-06   4.617 3.89e-06 ***
## X6              1.753e-03  9.794e-05  17.895  < 2e-16 ***
## X7              1.008e-05  1.868e-04   0.054 0.956963
## X8              9.847e-02  1.495e-03  65.876  < 2e-16 ***
## X10             7.108e-05  1.330e-05   5.346 8.97e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.02786103)
##
##     Null deviance: 15220  on 541199  degrees of freedom
## Residual deviance: 15078  on 541184  degrees of freedom
## AIC: -401904
##
## Number of Fisher Scoring iterations: 2
```

{r eval=False}

```
skuinformation=read.csv('train_skuinformation_missingvalue_removed.csv')
cuinformation=read.csv('train_vcuinformation.csv')
skutest <- read.csv('test_skuinformation_missingvalue_removed.csv')
vcutest <- read.csv('test_vcuinformation.csv')
names(cuinformation)[11:70]=as.character(skuinformation[1:60,1])
names(vcutest)[11:70]=as.character(skutest[1:60,1])
library(tidyr)
temp=pivot_longer(cuinformation,c(11:70))
temp1 <- pivot_longer(vcutest,c(11:70))
skuinformation=as.data.frame(skuinformation)
skutest <- as.data.frame(skutest)
temp=as.data.frame(temp)
temp1 <- as.data.frame(temp1)
names(temp)[11]="sku_ID"
names(temp1)[11]="sku_ID"
train=merge(temp,skuinformation, by = c("date", "sku_ID"))
test=merge(temp1,skutest, by = c("date", "sku_ID"))
train1 <- train
library(car)
```

```
## Loading required package: carData
```

```
logistic <- glm(value~.-date-user_ID-sku_ID-brand_ID, data = train)
vif(logistic)
```

```
##     user_level           plus         gender            age marital_status
##       1.611299       1.226179       1.135888       1.216860       1.291449
##      education     city_level purchase_power             X1             X2
##       1.369711       1.032982       1.040994       1.092437       1.153917
##             X3             X4             X5             X6             X7
##      31.936914      39.831991       1.571821       1.267967       1.128033
##             X8             X9            X10
##       1.142353      63.157328       1.216328
```

```
logistic1 <- glm(value~.-date-user_ID-sku_ID-brand_ID-X9-X4-X3, data = train)
vif(logistic1)
```

```
##     user_level           plus         gender            age marital_status
##       1.611299       1.226179       1.135888       1.216860       1.291449
##      education     city_level purchase_power             X1             X2
##       1.369711       1.032982       1.040994       1.021031       1.071152
##             X5             X6             X7             X8            X10
##       1.141170       1.046736       1.119958       1.048919       1.174647
```

```
summary(logistic1)
```

```
##
## Call:
## glm(formula = value ~ . - date - user_ID - sku_ID - brand_ID -
##     X9 - X4 - X3, data = train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -0.107  -0.006  -0.003   0.001  38.908
##
```

```
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -6.184e-03  2.205e-03  -2.805 0.005039 **
## user_level     -2.274e-04  3.166e-04  -0.718 0.472654
## plus            3.352e-04  5.040e-04   0.665 0.505981
## gender         -2.765e-04  2.824e-04  -0.979 0.327531
## age            -2.043e-05  2.875e-05  -0.710 0.477427
## marital_status  5.039e-04  5.437e-04   0.927 0.354042
## education      -1.628e-04  4.705e-04  -0.346 0.729389
## city_level     -2.949e-04  2.160e-04  -1.365 0.172152
## purchase_power -1.369e-03  4.881e-04  -2.805 0.005024 **
## X1              3.548e-03  2.171e-04  16.343  < 2e-16 ***
## X2             -3.091e-04  9.261e-05  -3.338 0.000844 ***
## X5              4.240e-05  9.182e-06   4.617 3.89e-06 ***
## X6              1.753e-03  9.794e-05  17.895  < 2e-16 ***
## X7              1.008e-05  1.868e-04   0.054 0.956963
## X8              9.847e-02  1.495e-03  65.876  < 2e-16 ***
## X10             7.108e-05  1.330e-05   5.346 8.97e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.02786103)
##
##     Null deviance: 15220  on 541199  degrees of freedom
## Residual deviance: 15078  on 541184  degrees of freedom
## AIC: -401904
##
## Number of Fisher Scoring iterations: 2
```

Appendix - CART

merge train(delete)

```
library(rpart) # CART
library(caret) # cross validation
library(lattice)
library(tidyr)
skuinformation <- read.csv('train_skuinformation_missingvalue_removed.csv')
cuinformation <- read.csv('train_vcuinformation.csv')
delete <- c()
for(i in 1:dim(cuinformation)[1]){
  if(sum(cuinformation[i, 11:70]) == 0){
    delete <- c(delete, i)
  }
}
cuinformation <- cuinformation[-delete, ]
# remove the first column of skuinformation
skuinformation <- skuinformation[-1]
names(cuinformation)[11:70] <- as.character(skuinformation[1:60, 1])
temp <- pivot_longer(cuinformation, c(11:70))
skuinformation <- as.data.frame(skuinformation)
temp <- as.data.frame(temp)
names(skuinformation)
names(temp)[11] <- "sku_ID"
names(skuinformation)[3] <- "date"
skuinformation[, 'sku_ID'] <- as.character(skuinformation[, 'sku_ID'])
train <- merge(temp, skuinformation, by = c('date', "sku_ID"))
```

merge test

```
skuinformation_test <- read.csv('test_skuinformation_missingvalue_removed.csv')
cuinformation_test <- read.csv('test_vcuinformation.csv')
# remove the first column of skuinformation_test
skuinformation_test <- skuinformation_test[-1]
cuinformation_test <- cuinformation_test[-71]
delete.test <- c()
for(i in 1:dim(cuinformation_test)[1]){
  if(sum(cuinformation_test[i, 11:70]) == 0){
    delete.test <- c(delete.test, i)
  }
}
cuinformation_test <- cuinformation_test[-delete.test, ]
names(cuinformation_test)[11:70] <- as.character(skuinformation_test[1:60, 1])
temp_test <- pivot_longer(cuinformation_test, c(11:70))
skuinformation_test <- as.data.frame(skuinformation_test)
```

```
temp_test <- as.data.frame(temp_test)
names(temp_test)[11] <- 'sku_ID'
names(temp_test)[1] <- 'date'
names(skuinformation_test)[3] <- "date"
skuinformation_test[, 'sku_ID'] <- as.character(skuinformation_test[, 'sku_ID'])
test <- merge(temp_test, skuinformation_test, by = c('date', "sku_ID"))
```

CV

```
CV <- function(k, observation, seed){
  cvlist <- list()
  set.seed(seed)
  n <- rep(1:k,ceiling(observation/k))[1:observation]
  temp <- sample(n,observation)
  x <- 1:k
  seq <- 1:observation
  cvlist <- lapply(x,function(x) seq[temp==x])
  return(cvlist)
}
k <- 10
observation <- nrow(train)
seed <- 100
cv <- CV(k, observation, seed)
```

The train and test set of every items

```
num.train <- length(unique(train$sku_ID))
train.item <- vector('list', num.train)
items <- levels(factor(train$sku_ID))
for(i in 1:dim(train)[1]){
  for(j in 1:num.train){
    if(train$sku_ID[i] == items[j]){
      train.item[[j]] <- rbind(as.data.frame(train.item[[j]]), train[i, ])
      break
    }
  }
}

num.test <- length(unique(test$sku_ID))
test.item <- vector('list', num.test)
for(i in 1:dim(test)[1]){
  for(j in 1:num.test){
    if(test$sku_ID[i] == items[j]){
      test.item[[j]] <- rbind(as.data.frame(test.item[[j]]), test[i, ])
      break
    }
  }
}
```

```
for(i in 1:60){
  train.item[[i]] <- train.item[[i]][, -c(13, 3, 2 ,1)]
  train.item[[i]]$value <- ifelse(train.item[[i]]$value == 0, 0, 1)
  train.item[[i]]$value <- as.factor(train.item[[i]]$value)
}
for(i in 1:60){
  test.item[[i]] <- test.item[[i]][, -c(13, 3, 2 ,1)]
  test.item[[i]]$value <- ifelse(test.item[[i]]$value == 0, 0, 1)
  test.item[[i]]$value <- as.factor(test.item[[i]]$value)
}
```

price

```
order <- read.csv('JD_order_data-2.csv')
order <- order[, c('sku_ID', 'final_unit_price')]
price <- data.frame('sku_ID' = NULL, 'price' = NULL)
sku <- unique(train$sku_ID)
for(i in 1:60){
  stock <- order[order$sku_ID == sku[i], ]
  price[i, 1] <- sku[i]
  price[i, 2] <- mean(stock$final_unit_price)
}
names(price) <- c('sku_ID', 'price')
```

```
list <- list()
```

CART(after delete)

```
for(j in 1:60){
  train_p <- train.item[[j]]
  test_p <- test.item[[j]]
  fn <-  max(price[j, 2], 1)
  fp <- (sum(as.numeric(train_p$value) - 1)/length(train_p$value)) * fn
  if(fp == 0){
    fp <- 1
  }
  cost <- matrix(c(0, fn, fp, 0), 2, 2)
  if(sum(test_p$value == 1) != 0){
    accuracy <- matrix(rep(0, 10*10), 10, 10)
    k <- 10
    observation <- nrow(train_p)
    seed <- 100
    cv <- CV(k, observation, seed)
    if(j != 7 & j !=15 & j != 27 & j != 31){
      for(m in 1:10){
        for(n in 1:k){
          train.cv <- train_p[-cv[[n]], ]
          val.cv <- train_p[cv[[n]], ]
          mod.cv <- rpart(value ~., data = train.cv, method="class",
                          parms=list(split='information', loss = cost),
```

```
                    # information here stands for information gain,
                    # which means we choose cross-entropy as our split rule.
                    minbucket=5, cp = 0.001 * m)
        predict2 <- predict(mod.cv, newdata = val.cv, type = "class")
        conf <- table(val.cv$value, predict2)
        acc <- (conf[1, 1] + conf[2, 2])/sum(conf)
        accuracy[m, n] <- acc
      }
    }
    acc.max <- max(apply(accuracy, 1, mean))
    cp <- which.max(apply(accuracy, 1, mean)) * 0.001
    # Find the model
    predict <- matrix(1:(10 * dim(test_p)[1]), ncol = 10)

    # construct the model
    mod <- rpart(value ~. , data = train_p, method="class",
                    parms=list(split='information', loss = cost),
                    # information here stands for information gain,
                    # which means we choose cross-entropy as our split rule.
                    minbucket=5, cp = cp)
    predict.tst <- predict(mod, test_p, type="class")
    conf <- table(test_p$value, predict.tst)
    # TPR
    TPR <-  (conf[2, 2])/sum(conf[2, ])
    correct <- conf[2, 2]
    sum_1 <- sum(conf[2, ])
    # FPR
    FPR <- (conf[1, 2])/sum(conf[1, ])
    # ACC
    ACC <- (conf[2, 2] + conf[1, 1])/sum(conf)
    result <- data.frame('TPR' = TPR, 'FPR' = FPR, 'ACC' = ACC, 'correst' = correct, 'sum_1' = sum_1)
    list[[j]] <- result
  }
 }
}
result_cart <- list
```

```
W <- c()
for(i in 1:60){
  a <- train.item[[i]]
  v <- as.numeric(a$value)
  w <- sum(v)
  W <- c(W, w)
}

W <- W/(sum(W))
acc.vec <- c()
tpr.vec <- c()
for(i in 1:60){
  acc <- list[[i]]$ACC
  tpr <- list[[i]]$TPR
  if(is.null(acc) == TRUE){
    acc <- 0
  }
```

```
  if(is.null(tpr) == TRUE){
    tpr <- 0
  }
  tpr.vec <- c(tpr.vec, tpr)
  acc.vec <- c(acc.vec, acc)
}
sum(W * t(acc.vec))
sum(W * t(tpr.vec))
```