Analysis_Report

Recommendation_Engine_TensorRec

Data Processing:

This project has used 1.Customer, 2.prod_cat_info, and 3.Transaction data. The transaction data (Transactions.csv) serves as a collaborative feature, because it describes the user's (or customer's) transaction-what and when they purchased, while the customer (Customer.csv) and product information (prod_cat_info.csv) serve as content features and item features respectively and performed different data cleaning techniques to trace and filter the outliers in the datasets.

Training Models and Methodology:

Overview:

Recommendation system is increasing in popularity — every data analyst, data scientist, and data engineer in retails and consumer-related business; whether in e-commerce or traditional offline business will come across these inevitable and important applications of Machine Learning. Perhaps it is not an exaggeration to acclaim it as one of the most high-impact products of ML. E-commerce has benefited a lot from machine learning prediction in improving business operation and increasing sales return.

Types of Recommendations:

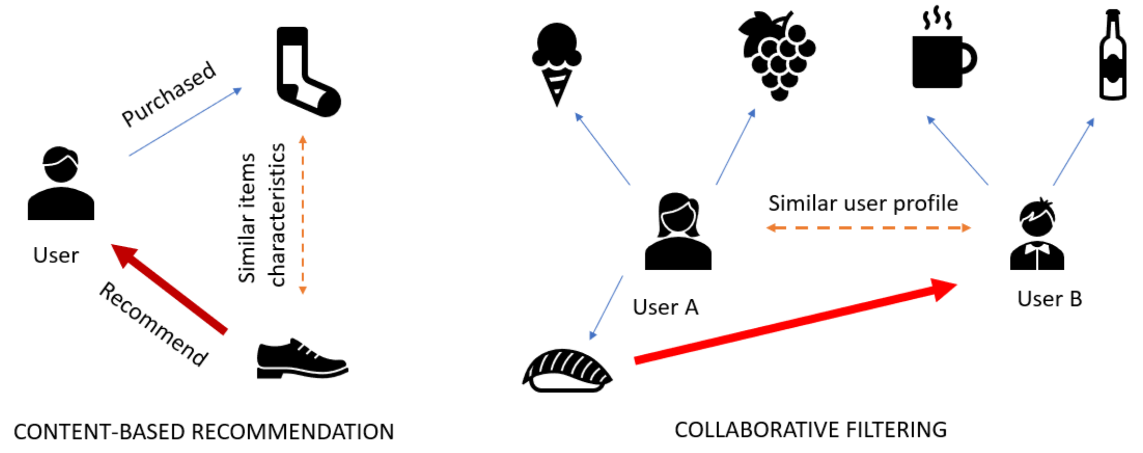
In retail, there are two kinds of recommendation commonly used, which are:

1. Content-based recommendation:

This system uses item's explicit features to represent interaction in between them. For example, if a user has purchased an item (e.g. a pair of socks), then the algorithm will recommend a similar or relevant item (e.g. shoes).

2. Collaborative Filtering:

This system is based on the idea "User who likes X also likes Y", where user's historical transaction in between these two items is chained together in building relationships among other users. If user X had purchased milk, user Y, who is similar to user X but have not yet purchased this item, will receive a recommendation to buy it.

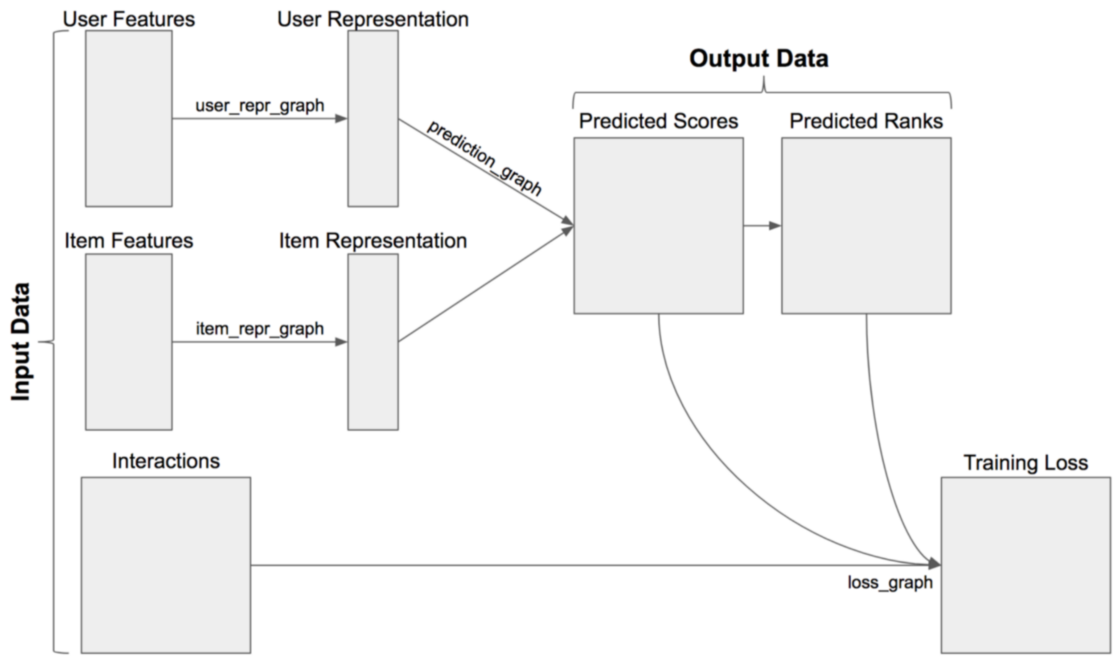


Two-Popular types of recommendations in Retail Industry

A combination of both approaches, with the help of TensorRec we will demonstrate in this project, is simply a hybrid of them, processes both user similarity and item similarity altogether. Recommendation engine may utilize collaborative filtering, content-based filtering, or a hybrid of both.

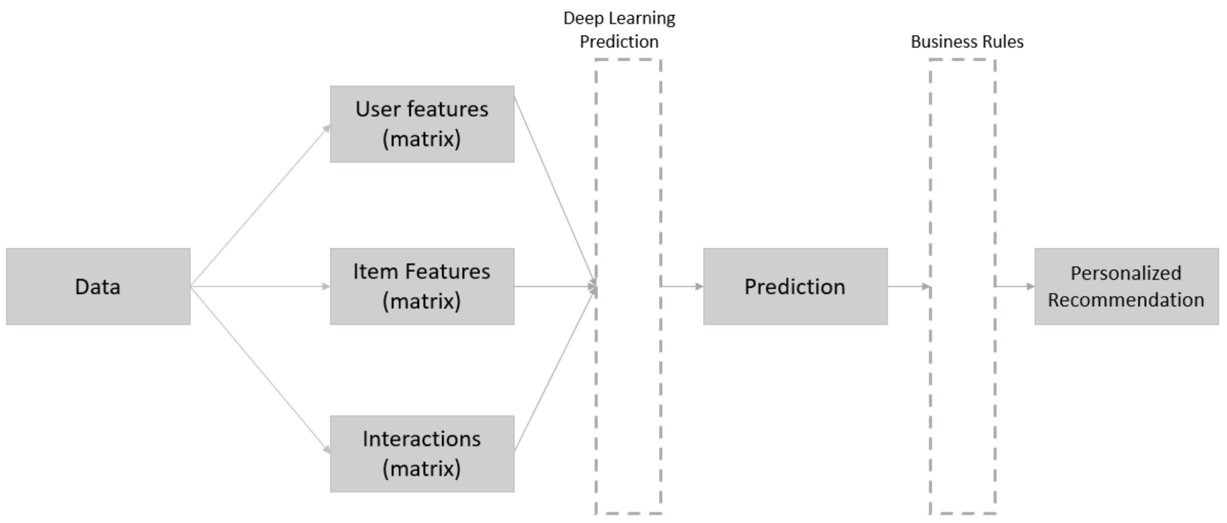
The transaction data (Transactions.csv) serves as a collaborative feature, because it describes the user's (or customer's) transaction what and when they purchased, while the customer (Customer.csv) and product information (prod_cat_info.csv) serve as content features and item features respectively.

Features can be either explicit or implicit, visible or hidden, either at face value or derived through some feature engineering. As a data scientist, feature engineering is an essential part of data wrangling, almost necessary in every ML algorithm, this recommendation system nonetheless. To achieve this, we will perform some unsupervised ML and some data wrangling on the transaction data to derive new features.



TensorRec_Input_Output_Data-Flow_Diagram

The key to TensorRec is to realize that there are three important components User Features, Item Features, and Interactions. As shown in the diagram above, the input data consists of user and item features, and interaction. The engine takes these three data to build a model that ranks the relevant interaction for each user.



Data-Science_General_Pipeline_Recommendation-System

We can take advantage of the powerful Deep Learning backend of TensorRec that processes training and testing the prediction. We only need to ensure we supply clean and relevant information into feature matrices, and any business case (e.g. eliminate long-tailed product, recommend only to active customers, customized category basket etc.) may be added after the prediction is done via data filtering.

The customer data, which is helpful for us to identify customer features. Is this customer information sufficient? , or is there any additional information that we can derive from the data? Remember that feature engineering is crucial to every successful ML algorithm, and this project is no different. Since TensorRec does not define the requirement of item features and customer features, we can take advantage of this liberty by using feature engineering to derive enhanced item and customer features. It can be as simple or as complex as we want, and so we opt for a twist and use a common marketing tool named “RFMV” as an additional feature of customers.

RFMV — Marketing Analytics into Machine Learning

Thousands of products. Thousands of customers. Which cereal box should I get?

A goal of feature engineering is to derive features not explicitly present in the data but to derive hidden patterns or correlation that may exists in between them. To achieve this purpose, data scientist often utilizes unsupervised learning method such as K-Mean clustering and T-SNe, in hope to yield patterns, groups, or insights that are not explicitly visible at hand. Marketing analytics benefited from this application by performing customer segmentation using unsupervised clustering method. The application extends beyond marketing too; we can take advantage of this method as supplementary information to better understand our customers.

Among numerous metrics exists in retail and consumer goods industry, recency (R), frequency (F), and monetary (M) are often utilized in valuating customer. Marketers at times use variety (V) as well. Hence, we utilize all four metrics, abbreviated RFMV in our customer feature analysis.

1. RECENCY (R):

Recency is defined as how recent a customer purchases an item from the current date (now date).

In this case, measured in days. A customer that has not engaged with us for a long time, say, a year, may need a business continuity and development evaluation.

Note: Here RECENCY is measured in days.

2. FREQUENCY (F):

Frequency denotes how often a customer purchases from us/retail_store.

A high-value customer may purchase with us/retail_store frequently.

Note: Here FREQUENCY is measured in number of times.

3. MONETARY (M):

Monetary, as the word implies, shows how much the customer had paid for the purchases.

Note: Here MONETARY is measured in the term of money/amount/price.

4. VARIETY (V):

Variety denotes the different types of item he/she has purchased.

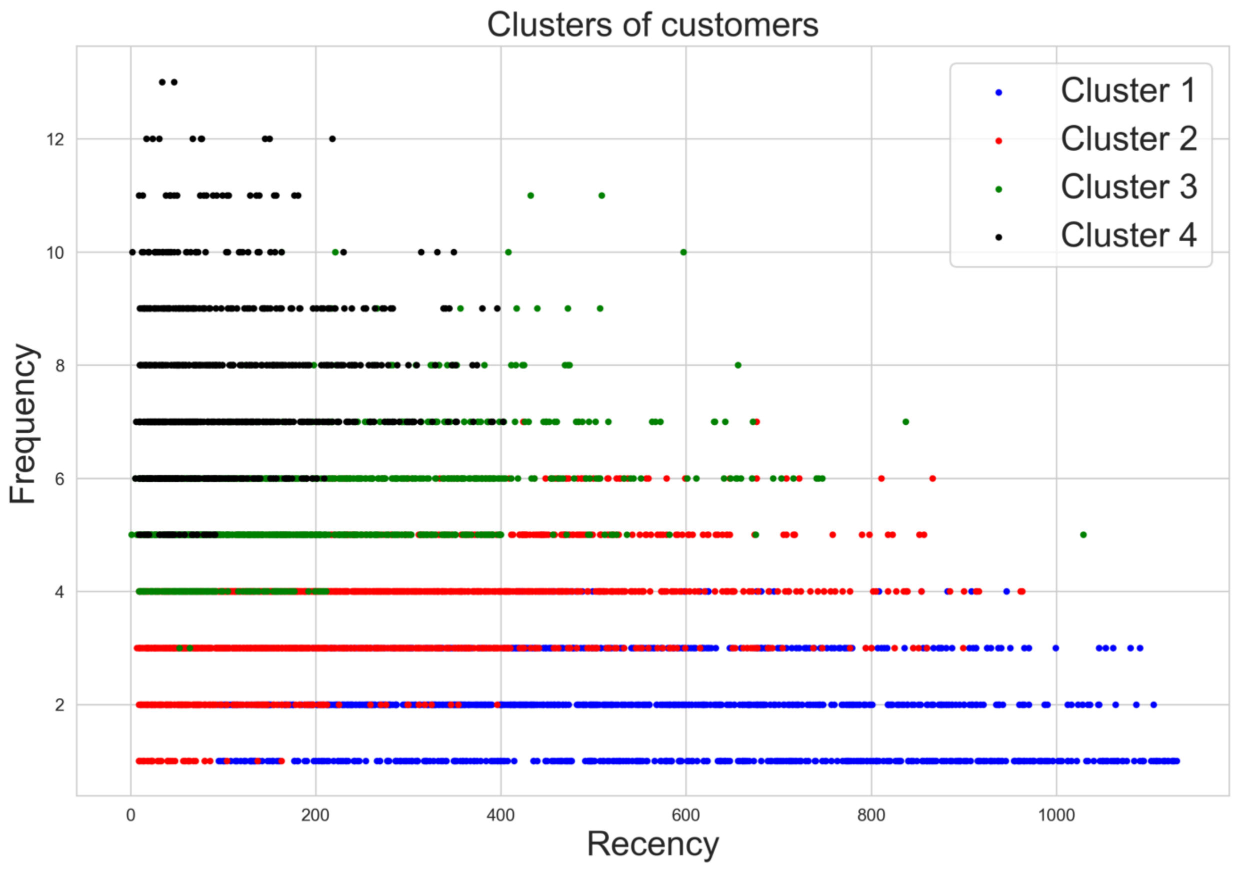
Note: Here VARIETY is measured in number of types.

Clustering Customers:

Clustering gives us a simplified window to see customer’s characteristics that is otherwise rather difficult to be visualized or comprehended by these four different segments. Not only it provides us with simplified customer metrics, it also adds another spectrum of information to the recommendation algorithm without overloading it with too much dimension (in this project, only two features as opposed to four).

Optional: Plot the two components with their respective RFMV to see the shape of the clusters

Optional: We can try to plot other RFMV components too for predictive analysis.



Clusters of Customers using Recency and Frequency

We can visualize the clusters to see how it spreads out within Recency and Frequency. Notice that certain clusters occupy certain range, for instance, blue customers entail medium-to-high Recency but low Frequency. Refer my GitHub repo on how we perform the clustering method and plotting the chart.

Hyperparameters:

The Recommendation Engine:

Different businesses may assert different business case. Some businesses may want to promote long-tailed SKUs (Stock Keeping Units), while others may want to regain lost or inactive customers. Depending on the business objective, you may want to use this algorithm for a variety of scenario. You could be recommending to active, high-potential customers (they are the “low-hanging fruits”) or you could revive long-lost customers (rather difficult but possible). In this example, we will apply the first scenario, which is, we want to recommend to active customers, and the easiest way to determine that is to filter out customers with recency beyond 360 days (one year). We have the data for that, and all we need to do is to filter them out. This is optional, as you can also simply recommend to everyone, buy you may risk losing accuracy as the data may get convoluted with noise rather than relevant info.

TensorRec accept matrix, and not dataframe, therefore we need to transform them into matrix, now we should transform the important data to be passed to the interaction matrix, user feature, and item feature.

We have three (3) features as required by TensorRec:

1. Interaction Matrix:

Getting the Customer's transaction for every material in terms of sales quantity.

2. User Features Matrix:

There are two features that are used to build the User Feature Matrix here:

1. First the customer's unique categories they buy.

2. Second the clusters they belong to.

3. Item Features Matrix:

We use different categories of items as the feature to build the Item Feature Matrix here.

Create COO_Matrices using Scipy function:

1. COO_Matrices are sparse matrices, mostly filled with zero.

2. It speeds up the processes and saves a lot of memory.

Using Scipy’s *coo_matrix* function, we transform the user, item, and interaction features into sparse matrices. Using sparse matrices, especially for large data set, is always a good step as it reduces computing complexity, especially when the matrices are to be fed into Neural Network, which itself a heavily complex algorithm and often consumes a huge amount of time in calculating the forward and backward propagation.

In TensorRec we can conveniently pass the user feature, item feature, and the interaction as a one-line code into the model (whichever model you deem as best fit into the data set).

We can take a step further and split the data into train and test set. As with any good Machine Learning model, we should perform the split conventionally at 80–20 ratio. How can we do it efficiently and effectively here? We can use a method called masking, where we mask at random 20% of the interaction data. The mask hides the interactions, so it yields as if the customer never purchases the item before, while separating the actual transaction of those masked item in the test set.

TensorRec offer several Neural Network representation graphs, based on the order of complexity and depth of calculation required by the data set. Always believe that there is no single blanket algorithm that works for all data set; one graph may fit on a certain type, while others may work on a different kind. So in this project we use:

1. DeepRepresentationGraph for user representation,

2. NormalizedLinearRepresentationGraph for item representation,

3. WMRBLossGraph for loss.

Try several different graphs to find the best fit for the model. In addition, we must tune the model by trying different parameters and representation graph in order to find the best fit. The best practice here is to combine and mix different representation graph for user features, item features, and loss graph; the choices above fits this case, but we can try others, which some of them are listed in TensorRec Git page.

Testing: Recall at K:

After we have ran the model, we should proceed by testing if this model is the best fit for the data. But what do I mean by best fit? What is the best fit for a recommendation engine? As we have discussed above, recommendation engine prescribes several outputs that are relevant to the user. The key here is *relevant*. In retail, relevant items means that users are likely to buy it if recommended, regardless whether the items are closest to his past purchases or to his related user's past purchases. We do not know what is in user's mind, whether he likes it or not, but we can assume that the user has some affinity to it, or in other words, can *relate* to that item. Remember, businesses only care about selling recommended product, but how can we ensure the recommended user will buy (or most likely to buy) the item?

One way we can ensure the generated recommendation is *relevant* is to obtain some degree of accuracy. In most (if not all) ML, the higher the accuracy, the better the algorithm. Similar metric applies in recommendation system, but there is a catch.

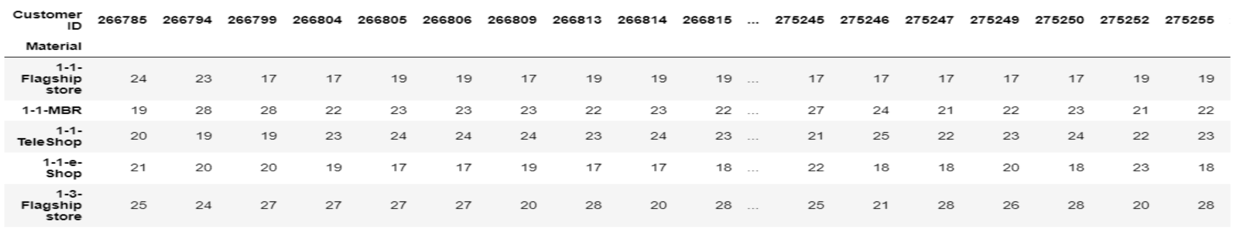
In recommendation system, two types of metrics used frequently are precision@k and recall@k, which conventionally defined as:

Precision:
(The percentage of recommended items @k that are relevant) / (Total # of recommended items)

Recall:
(The percentage of recommended items @k that are relevant) / (Total # of relevant items)

These definitions ask for relevance — but how do we define it? What does a relevant item mean exactly to a customer? This is a problem greatly brought forward by Joseph Konstant, where he stated that there is no ground truth for relevancy - no recommendation system has the base truth about relevance. Kirill Alexandrovich also discusses this problem in greater details here.

So, ideally, if the customer had purchased 50 items in the past, and the recommendation has successfully recommend 40 out of those 50 items, where 40 items were purchased by the customers before, then the recall metrics (assuming total items sold by the business is 50) is 40/50, which is 80%. That is not too bad. But think, would not it be better if the accuracy is 100% that means, if the recommendation is able to recommend all the products that the user has purchased? If we want to recommend new products that the customer has not purchased before, then we would not be able to fit these new items into the ranking, because the ranking is filled with only purchased items, leaving non-purchased items out of the equation. The gap, 20% loss from the accuracy, provides some rooms for items that can be recommended but yet to exists in a customer's existing portfolio. That is why achieving 100% recall accuracy is probably not a practical solution.



Customer ID	266785	266794	266799	266804	266805	266806	266809	266813	266814	266815	...	275245	275246	275247	275249	275250	275252	275255
Material																		
1-1-Flagship store	24	23	17	17	19	19	17	19	19	19	...	17	17	17	17	17	19	19
1-1-MBR	19	28	28	22	23	23	23	22	23	22	...	27	24	21	22	23	21	22
1-1-TeleShop	20	19	19	23	24	24	24	23	24	23	...	21	25	22	23	24	22	23
1-1-e-Shop	21	20	20	19	17	17	19	17	17	18	...	22	18	18	20	18	23	18
1-3-Flagship store	25	24	27	27	27	27	20	28	20	28	...	25	21	28	26	28	20	28

Personalized ranking of items (row index) for each customer (column)

So here we go, we accept the model at around 90% accuracy. We have yielded a set of ranking of personalized relevant products for each user, shown on the table above. Each material is ranked from 1 to *n* (*n* = total # of items) differently for each customer. If the system desires to produce ranking for items that they never purchased before, we can simply filter the output to only display that.

Final Note:

Very often, online platform recommendations are putting products that very often make no sense to the user. The problem may lie in the engine, but more often it lies in the input data itself - what types of features and labels we have inserted into the engine, and the output data - how the output will look like at receiver's end? These are the types of data wrangling and manipulation that we need to consider, adds, and modify. Think, if we want a clear distinction in between white chocolate and dark chocolate, then the item features should have categories that clearly show the difference in between these two. However, if we do not carry much variety of chocolate, or the difference does not matter, then perhaps they may be lumped together with other products into one category, say, confectionery. These types of consideration are all depend on the business context and request, requiring us to run the engine several times with different features, rather than simply altering and tuning the parameters.

Often, recommendation engines are expected to recommend "long-tailed" items, meaning that items that are previously not selling well, perhaps due to overshadowing from another competitor's product, lack of advertisement and marketing effort, or other factors related to market force. Some of the engines are also expected to yield "surprise" items — items that surprisingly never got popular but has almost-similar feature with items other customer had bought. Can we use recommendation algorithm to up-sell and push the long-tailed or hidden items to users? That is what many researchers and academicians have argued — that the system should be able to recommend items that users might not have otherwise know before. But how can we achieve that in TensorRec? The Neural Network is a black box process that is hard to be interpreted.

Analysis_Report

My suggestion is to apply workarounds on data manipulation and data wrangling instead of focusing too much on fine-tuning the algorithm. In addition, we should get close to Business Analysts, Data Engineers, SMEs (Subject Matter Experts) and Project Managers, understand the business context and different scenarios, and only then Data Scientists like us can deliver relevant results that truly add value to business.