A Market Basket Recommender System

Angéllica C. Araujo[1]

[1]Rio de Janeiro - RJ, Rio de Janeiro Brazil

angellica.c.a@gmail.com

1. Definition

1.1. Project Overview

The task of identifying correlations in sequential user shopping data is called Market Basket Analysis. The most common application of this prediction task is the indication of the next items that a user will be likely to consume. Therefore, the purpose of discovering a user's frequent buying patterns is to obtain information about the user's buying behavior [Kaur and Kang 2016].

Recommendation algorithms are also used to customize online stores, such as Amazon, which has a set of engineering models that can adapt a webpage content according to known user preferences [Amazon 2003]. Such computational tools play an important role in the industry, acting directly on the correlation between supply and demand. Although they belong to a thoroughly explored study area, these tools need to follow the transformations of the information inherent in their computational model.

Recent studies, such as that presented by Netflix, have already demonstrated the usefulness of exploring the association of different variables in recommendation results [Basilico and Raimond 2017]. This work is inspired by the recurring advances that machine learning techniques bring to different industrial areas. In particular, it aims to improve the results of a recommendation tool specifically directed to the retail and food sectors. The Kaggle challenge [Kaggle 2017] was also taken as inspiration and as a source of data input and metrics for comparison.

1.2. Problem Statement

Let $C = \{c^1, c^2, c^3, ..., c^n\}$ be a set of customers, $I = \{i^1, i^2, i^3, ..., i^n\}$ the set of itens and $E_c = \{E_c^1, E_c^2, E_c^3, ..., E_c^n\}$ the set of consumer events. A consumer event can be defined as $E_c^t, b \subset I, \forall c, t$ where for each set b bof consumed items there is a transaction that contains one or more items, for each user in a time frame t. An algorithm should consider a customer's consumption event history E_c^t and predict which items are most likely to be consumed, in addition to estimating the likelihood that a consumer will be interested in a given item. Thus, we can consider our problem as a classification modeling task.

We intend to reach this objective by following the sequence of steps described below:

1. Preprocess the set of orders and products to obtain enough informaion about Baskets b per customer
2. Train classification models considering the features related with the consumption events E_c of customer c
3. The classification task may consider different values for temporal features as input data

4. For each model produce as output a vector of probability of consumption per item
5. Analyze the results obtained for each model with evaluation metrics

With this proposed solution, we intend to demonstrate that different features, besides the item and user data, are important to consider in such a case. Indeed, they can generate some positive impact on the precision of the results produced by these computational tools when they are taken into account in a recommendation system. To this end, the recommendation system this work proposes will be composed with a classification model in order to perform the training and prediction tasks.

1.3. Metrics

We will use the F1 Score as proposed by the Kaggle challenge [Kaggle 2017]. Recall and precision fractions are considered in the calculation giving us a metric whose result is a balance between both fractions [Prentice-hall]. The general formula for F1 Score is:

$$F_\beta = (1 + \beta^2) \left(\frac{precision.recall}{(\beta^2.precision)+recall} \right)$$

The formula applied in this work is an adaptation known as F-score or F-measure. Taking $\beta = 1$, we have the following formula as basis for the calculation of score and validation of the results of the work:

$$F_1 = (2) \left(\frac{precision.recall}{precision+recall} \right)$$

Recall is a measure of proportionality analysis between positive outcomes and outcomes identified as positive. In this way, recall allows us to understand how complete the end result is. Precision allows us to analyze how accurate our result is, returning the ratio between results that are true positives and all results identified as positive (including false positives) [Prentice-hall].

2. Analysis

2.1. Data Exploration

The Instacart data set is a public release of the Instacart Online Grocery Shopping orders that was made available in 2017. The data set contains over 3 million orders made from more than 200,000 Instacart users. Each user has between 4 and 100 orders and each order has a sequence of the purchased products. Also, the week and hour of the day for each order is known, as is the relative time between them.

The provided dataset is a collection of .csv files. For this work, we will focus on the data provided by *order products* and *orders* files. The *order products* file specify which products were purchased in which order and the 'reordered' attribute indicates that the customer has a previous order that contains the product. We will use the *orders* file to select the set for training and testing in addition to building our baskets. [Kaggle 2017]. So, our work will receive as a input of orders the following data [Instacart 2017]:

orders (3.4m rows, 206k users):

- order_id: order identifier
- user_id: customer identifier
- eval_set: which evaluation set this order belongs in (prior, train or test)
- order_number: the order sequence number for this user (1 = first, n = nth)

- order_dow: the day of the week the order was placed on
- order_hour_of_day: the hour of the day the order was placed on
- days_since_prior: days since the last order, capped at 30 (with NAs for order_number = 1)

order products (prior, train and test) (30m+ rows):

- order_id: foreign key
- product_id: foreign key
- add_to_cart_order: order in which each product was added to cart
- reordered: 1 if this product has been ordered by this user in the past, 0 otherwise

2.2. Exploratory Visualization

The exploratory analysis we will carry out in the data corresponds to the distribution of the categorical variables *order_dow* and *order_hour_of_day*. The figure 1 illustrates the distribution of the *order_dow* variable values in the dataset.

Figure 1. Volume of orders by week day

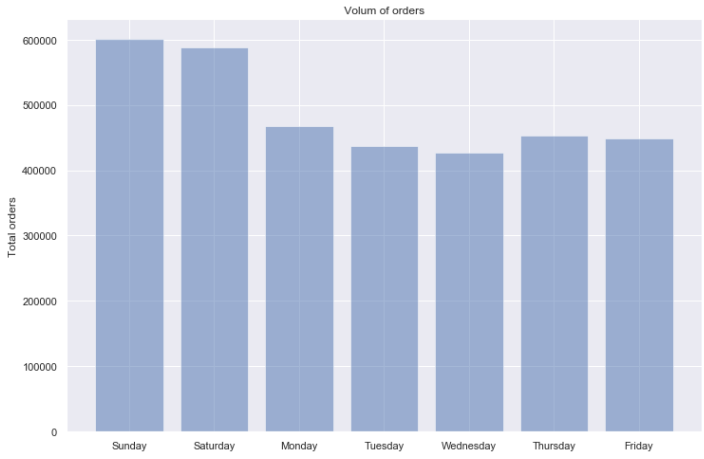
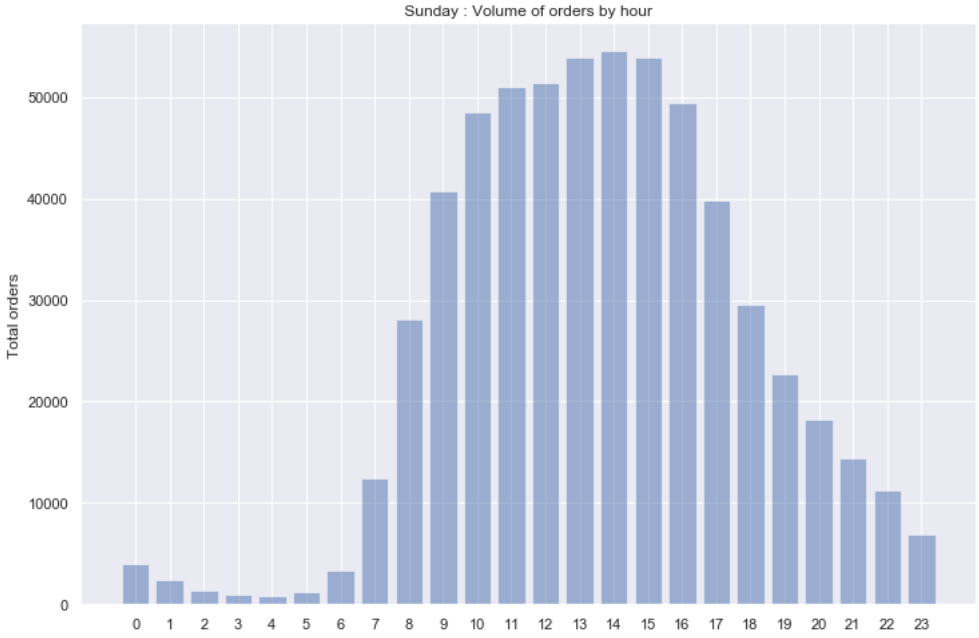


Figure 1 ilustrates a subtle difference in frequency for the *Sunday* and *Saturday* categories values. Given the nature of the problem we are working on, it is expected that there will be a greater frequency of the two categories, Sunday and Saturday when it is assumed that consumers have more free time on weekends to make purchases.

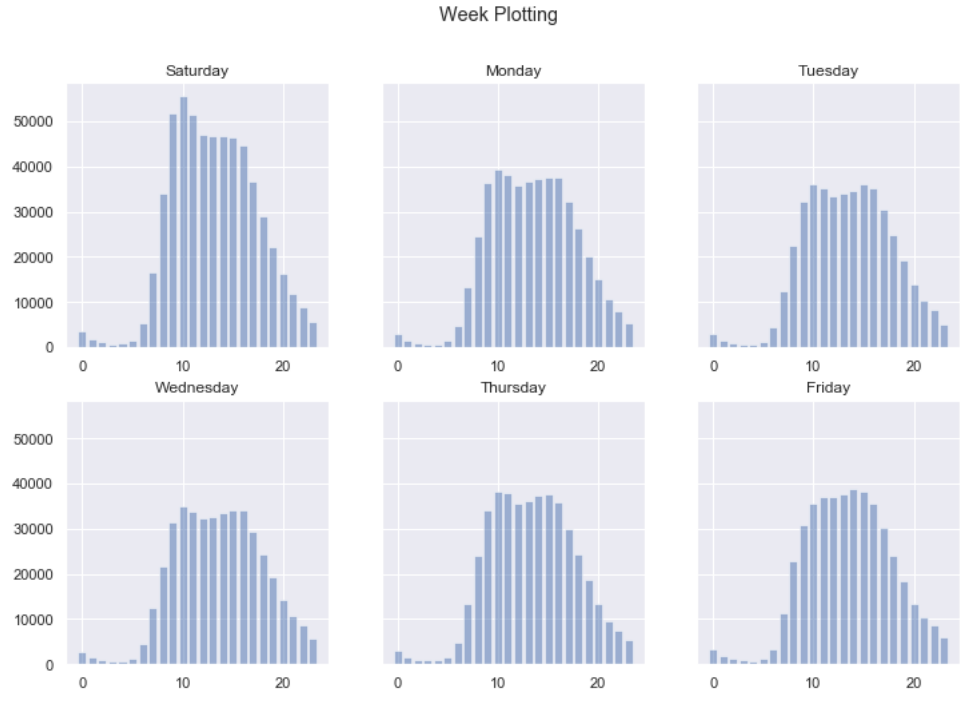
The second step of the exploratory analysis is performed on the *order_hour_of_day* variable. Figure 2 shows the frequency of purchases per hour of the day. Figure 3 shows the same analysis performed in relation to the other values of *order_dow*.

Figure 2. Volume of orders by week day: Sunday



The *Sunday* category concentrates the highest frequency of data, and is therefore highlighted in the set of distribution charts. Figure 2 shows a high sales volume between the range *9* and *17* indicating that, on weekends, both morning and afternoon periods concentrate high values of sales occurrences.

Figure 3. Week view of the volume of orders



At this point, we need to check whether the pattern stays the same throughout the

week. Despite the low frequency of the data on other days of the week, it can be seen that the high pattern in the period between *9* and *17* remains consistent. This information is relevant because a variation in this pattern could imply the identification of outliers and negatively impact the model.

2.3. Algorithms and Techniques

A classification model will be applied for this work. Our intention is to find a routine that produces a vector of probabilities on the items contained in the set I as a result. The main objective is to calculate $Pr(c$ being interested in i in a time frame $t_c + 1|E_c)$.

Once we have formulated a hypothesis, we can use of mathematical tools to predict occurrence probabilities. This is true because we are dealing with a classification problem that works with prior knowledge to evaluate the probability of our hypothesis [Science 2018].

As input, our solution receives a consumption event E_c, defined by a sequence of chosen items. Based on this, we will test results of a Naive Bayes Classifier and a Random forest model to predict the next most likely item to be chosen by the consumer c, complementing the shopping basket like a recommender system [Science 2018].

The Random Forest algorithm has some characteristics which are relevant to the problem we are studying, when compared to other classification algorithms. With the Random Forest algorithm, it is possible to predict whether a result belongs to a certain class through a probability value. Moreover, because it is based on a decision tree, the model is barely susceptible to overfitting.

In order to obtain the Random Forest algorithm's best performance, it is necessary, even before executing the model, to perform feature engineering and seek to increase the training set with higher quality data. Last but not least, model's parameters will be tuned for optimizing the classification model.

Hyperparameters are values that are set only before the training phase. In this case, we will work on the number of decision trees in the forest and the number of features used to split each leaf node.

- n_estimators
- min_samples_split
- min_samples_leaf
- max_features
- max_depth
- bootstrap

2.4. Benchmark

Because we choose to use the Kaggle competition's success parameters, this work will use the value of 0.4091449 as the ideal maximum, inspired on Kaggle's Leaderboard panel. We will consider to be acceptable any result in the interval between 0.4074450 and 0.4047891, referring to the fifty best results of the competition [Kaggle 2017].

As a second step, we will compare the answers obtained by the two classifiers. The objective here is to verify which of the two models will perform better, producing

more accurate results. For example, we can structure our Naive Bayes classifier based on features that describe a rule that a consumer will decide buy the product i and then compare the results with the Random Forest model which works with a random selection of features based on prior occurrences [Science 2018].

3. Methodology

3.1. Data Preprocessing

Data Preprocessing is a general approache for improving machine learning models tike the random forest model. In this section, we will explain how feature engineering was used to improve the average results for *Precision*, *Recall* and *F1-Score* [Koehrsen 2018].

- Perform feature engineering for extracting *User Behavior* information
- Perform data preprocessing for products set
- Combine new features as a *Baskets* dataset
- Apply *Baskets* into models for training

In the Exploratory Visualization section, we found that there is a pattern in shopping frequencies throughout the week, with a higher frequency of purchases registered over the weekend. Since there are significant values for the purchasing attributes, we begin by counting the continuous data related to consumer interest for a product in a general perspective, as follows:

- total_orders: total orders per consumer
- count_items_on_cart: total items in cart per consumer
- general_item_reorder_freq: the frequency of purchase per product

The training set is based on the set of previous orders of the consumers, that is, the orders with the classification *eval_set* equals *prior*. Before calculating the values for the above features, it was necessary to combine other data sets in order to to perform the necessary calculations. Thus, the first sequence of data grouping was:

- group *order_products* set with *orders* set on *order_id* attribute for both *prior* and *train* values of *eval_set*
- group *order_products prior* set with *products* set on *product_id* attribute

The *pandas* library was used to carry out the pre-processing steps we previously described. Below, we demonstrate part of the routine implemented for the data grouping of orders, users and products.

```
import pandas

# Load data
orders = pandas.read_csv("orders.csv")
order_products_train = pandas.read_csv("order_products__train.csv")
order_products_prior = pandas.read_csv("order_products__prior.csv")
products = pandas.read_csv("products.csv")
```

The process of grouping the data sets was done through the merge and concat methods of the Pandas library. This way, we were able to group the data from the relationship criteria. We also managed to gather the data without prior criteria, such as a concatenation of data.

```
orders_products_train =
pandas.merge(orders[orders['eval_set']=='train'], order_products_train,
    on='order_id',
    how='right')

orders_products_prior =
pandas.merge(orders[orders['eval_set']=='prior'],
    order_products_prior,
    on='order_id',
    how='right')

frames = [orders_products_train, orders_products_prior]
orders_products = pandas.concat(frames)
```

The *order_products* set gives us important information about whether a product has already been ordered more than once by the consumer through the *reordered* attribute and allows us to know the quantity of items in a cart with the *add_to_cart_order* attribute. Information about the user were obtained with the *orders* set attributes.

The first part of the feature engineering process is to extract relevant information from the data set from a user or product purchase perspective. For example, we consider it relevant to know the frequency of product reorders for each user.

```
# 2.1 : Select reordered products
products_reordered = orders_products[orders_products['reordered']==1]

# 2.2 : Count reordered frequency for each user
count_items_on_cart =
    products_reordered
    .groupby(['user_id'])['add_to_cart_order']
    .aggregate('count')

count_items_on_cart =
    count_items_on_cart
    .reset_index(name='count_reorder_by_user')
```

The feature engineering process is not just about putting new features into the data set. After inserting the features mentioned above into the training set, we have dropped the following features: *eval_set*, *add_to_cart_order*, *reordered*, *days_since_prior_order*. These features have either been used to calculate the new features or have no relevant correlation with the data on consumer interest. Finally, Figure 4 shows the first ten rows of the training set.

Figure 4. First ten rows of the new training set

order_id	product_id	user_id	order_number	order_dow	order_hour_of_day	general_item_reorder_freq	count_items_on_cart	total_orders
1	49302	112108	4	4	10	101	9	3
1	11109	112108	4	4	10	3192	9	3
1	10246	112108	4	4	10	12498	9	3
1	49683	112108	4	4	10	67313	9	3
1	43633	112108	4	4	10	312	9	3
1	13176	112108	4	4	10	315913	9	3
1	47209	112108	4	4	10	170131	9	3
1	22035	112108	4	4	10	45639	9	3
816049	49302	47901	14	4	6	101	22	13
816049	22035	47901	14	4	6	45639	22	13
816049	20574	47901	14	4	6	7622	22	13

To see if we are going in the right direction, we extract a preliminary report before and after the change in the training set. Table 3.1 shows the first effect in the results after insertion of the new features. In all three indicators we obtained a gain of 0.03. In addition, we were able to improve accuracy from 0.644 to 0.668.

Table 1. Average and Total values before and after feature engineering

avg / total	Before	After
Precision	0.64	0.67
Recall	0.64	0.67
F1-Score	0.64	0.67

Despite the positive impact on early results, it is important to ensure that all data is being evaluated in the right way. For example, after some series of combinations between data sets and aggregation functions, we can not guarantee that the tuples have been correctly matched.

Thus, during the assembly phase of *user behavior*, we decided to perform a test with an user with ID equals to one, verifying that the total of the user purchase records remains the same until the moment of the formation of the data set used as input for the predictive models.

The variable *count_user_1_orders* quantifies the total of user 1 requests in the dataser *orders*. After the merges of the datasets *order_products* we need to ensure that there is no unnecessary duplication of purchase records. Therefore, we use the variable *count_user_1_orders_products* to calculate the total order of user 1 after the merges.

The figure 5 shows the values of each of the two variables. Therefore, we can verify that before and after the merge between data sets, the total of requests for user 1 has stayed the same.

Figure 5. Testing user 1 total of orders before and after dataset merge

count_user_1_orders	int	1	11
count_user_1_orders_products	Series	(11,)	Series object of pandas.core.series module

The expected result for the data set *User Bahavior* is a combination of general consumer information and consumer information that relates user and product. Therefore, for each user, we expect that there is a correlation between the number of records in *User Bahavior* and total orders. This is illustrated in figure 6, where we can see a little more than eleven records for user 1; this is the result we expected since *User Bahavior* a request assembles at least one or more products.

Figure 6. Testing User Behavior dataset for user 1

Index	user_id	count_reorder_by_user	count_orders_by_user	product_id	count_reorder_product_by_user
0	1	51	11	196	10
1	1	51	11	10258	9
2	1	51	11	12427	9
3	1	51	11	13032	3
4	1	51	11	13176	1
5	1	51	11	25133	8
6	1	51	11	26088	2
7	1	51	11	26405	2
8	1	51	11	38928	1
9	1	51	11	39657	1
10	1	51	11	46149	3
11	1	51	11	49235	2

The model training process will receive as input a set of data that is compound by the user‘s purchasing behavior set of data.

```
baskets = pandas.merge(general_user_behavior,
products_preferences,
on=['user_id'],
how='right',
validate='one_to_many')
baskets = pandas.merge(product_user_behavior,
baskets,
on=['user_id', 'product_id'],
how='right')
baskets = baskets.drop(['eval_set', 'order_number'], axis=1)
```

3.2. Implementation

The implementation process can be summarized as follows:

1. Training phase
2. Extraction phase of metrics
3. Phase of application

In this step, the classifier receives as input the output data from the preprocessing routine. The implementation of the training step was performed in the python file train.py. The routines implemented in the file can be divided into the following steps:

1. Import the necessary libraries containing the classifiers
2. Import the pre-processing methods and perform the routine as described in the previous section
3. Separate a portion of the data in training and test: this step is performed through the method *train_set_split*
4. Instantiate the Random Forest Classifier classifier
5. Train the classifier with the training data from step 3
6. Test the prediction on the test set obtained in step 3
7. Calculate performance metrics
8. If the metrics are unsatisfactory, check hyparameters and return to step 4

Below we present a preview of the code referring to steps 1,2,3 and 4. For implementation of the Random Forest Classifier classifier we chose the version provided by the *sklearn* library. The test set training set separation routine was also harnessed from same library. We chose the *sklearn* library because it allowed us to implement a consolidated version of the classifier as well as providing a code-readable syntax in the training phase, allowing us to focus on tasks such as validating results and improving them.

```
from sklearn.model_selection import train_test_split
# extract X_train and y_train
X_train, X_test, y_train, y_test =
    train_test_split(baskets_x, baskets_y, test_size=0.2)

# Random Forest Classification training
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 24,
                max_features= 6,
                random_state = 42)
classifier.fit(X_train,y_train)

# Use the resulting model to predict over the testing baskets
prediction_result = classifier.predict(X_test)
```

3.3. Refinement

In the refinement phase, we seek to improve the results obtained from the feature engineering process. We can consider the results obtained in the feature engineering process as good. However, we would like to get a little more impressed by it. In this work we optimize the random forest model with resources provided by the Scikit-Learn library.

The hyperparameter tuning process occurs before the training process of a model; that is, it is an activity the data analyst can perform. To determine the most appropriate values of the hyperparametres, we thus perform several tests, alternating the values between the hyperparameter in different combinations.

In the case of Random Forest, the hyperparameter configuration includes the total number of decision trees in the forest and the number of features considered ideal for each ideal tree when breaking one of its nodes. Because it is an experimental process operated

on the training data set, overfitting can occur. Hence, we decided to use a crossvalidation method available from Scikit-Learn to define the optimal ranges of hyperparameters.

```
{
    'n_estimators': 200,
    'min_samples_split': 5,
    'min_samples_leaf': 4,
    'max_features': 'auto',
    'max_depth': 10,
    'bootstrap': True
}
```

4. Results

4.1. Model Evaluation and Validation

At the end of the development process, a portion of the data set was separated for testing purposes. This process was performed after the feature engineering activities and before the hyperparameter tuning process. The final parameters for the Random Forest model were defined by methods available in the Scikit-Learn library, according to the small sample of the code, shown below:

```
from sklearn.model_selection import RandomizedSearchCV
RandomizedSearchCV(estimator = random_classifier,
                                param_distributions = random_grid,
                                n_iter = 100, cv = 3,
                                verbose=2,
                                random_state=42,
                                n_jobs = -1)
```

During the process of checking the first results, it was necessary to choose between one of the two selected models. In order to make the best decision, we use the first results presented in the classification report, before the tuning of hyperparametres, in the figures 7 and 8.

Figure 7. First classification report for 20k samples when training Random Forest Classifier

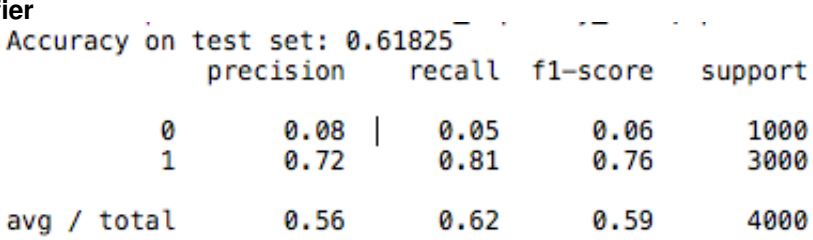


Figure 8. First classification report for 20k samples when training Gaussian Naive Bayes

```
Accuracy on test set: 0.60325
             precision    recall  f1-score   support

          0       0.33      0.68      0.45       937
          1       0.86      0.58      0.69      3063

avg / total       0.73      0.60      0.63      4000
```

4.2. Justification

We used the F1-Score formula, explained in the metrics section, to validate our Random Forest model and compare it to the Kaggle leaderboard results. Our proposal differs from Kaggle's challenge in some respects. For example, our model does not aim to respond to requests for an order that has already been made, but rather to highlight the product that is most interesting to a user.

The feature engineering and training process was performed in the Anaconda suite, using Spyder's resources, running on a MacBook Pro computer. Despite the available resources, it was necessary to perform a cut in the data set to make the study feasible, since at some point the kernel did not meet training for the millions of data in the original dataset.

Although the output of our method is not exactly identical to the one proposed in the Kaggle challenge, it is validated in a very similar profile; that is, checking if a product was requested again or not for a particular request. In our case, the unique identifier of the order is not part of the input set of the model because we carry out training on what we call consumer behavior and preferences.

The table below lists the first three models of the leaderboard in Kaggle and the F1-Score results of each of them. We tried to correlate the applied models but it was only possible to identify the model for the second place. Then, we added the score obtained by our model, after the feature engineering, before and after the improvement of the model.

Position	Kaggle Score	Kaggle Model	Random Forest Score	Tuned Random Forest Score
1	0.4091	NA	0.59	0.70
2	0.4082	XGBoost		
3	0.4081	NA		

5. Conclusion

5.1. Free-Form Visualization

The table below illustrates one of the Prediction consumption records of the Plain Bagelettes product for user 91. The final output of the result shows the probability values for association with each class. In this way, it focuses in the posterior criterion the best intervals of recommendation or not of a product when the values are many next to each other.

Product	User	Weekday	Hour	Not Recommended (%)	Recommended (%)
Plain Bagelettes	91	Friday	13	0.46	0.53

5.2. Reflection

The end-to-end problem solution can be summarized as follow:

1. Formally define the problem described previously in this document.
2. Search for articles and research support material.
3. Define language and library to be applied.
4. Choose IDEs for code implementation.
5. Structure the steps needed to solve the problem.
6. Define the models for the training phases.
7. Perform the activities inherent in solving the problem.
8. Collect results for evaluation of model improvements

Of course, the most complicated steps in the development of the project were those related to the preprocessing of features and the engineering of features, activities related to step 7. The Instacart data set is undoubtedly one of the most complete (if not the only) dataset when it comes to food-related purchasing information.

On the other hand, an amount of information this large must be handled with care. During the implementation of step 7, we performed occasional checks to ensure that the final data set used in the training did not present inconsistent or duplicate information.

5.3. Improvement

In order to improve the results of the F1-Score metrics, the training could be performed in one more portion of the data set if more hardware was used in the study of the problem. We propose the following hardware configuration for better execution of works with data sets such as those of InstaCart:

- Core i7 8700k
- 16gb RAM
- GeForce 1070ti
- Motherboard Asus Z370-F gaming

The better is the hardware configuration the better accurate will be model‘s results once one will not face problems running the model against a significant amount of data.

Reference

Amazon (2003). Amazon.com recommendations.

Basilico, J. and Raimond, Y. (2017). Déjà vu: The importance of time and causality in recommender systems. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, pages 342–342, New York, NY, USA. ACM.

Instacart (2017). The instacart online grocery shopping dataset 2017 data descriptions.

Kaggle (2017). Instacart market basket analysis.

Kaur, M. and Kang, S. (2016). Market basket analysis: Identify the changing trends of market data using association rule mining. *Procedia Computer Science*, 85:78 – 85. International Conference on Computational Modelling and Security (CMS 2016).

Koehrsen, W. (2018). Gathering more data and feature engineering.

Prentice-hall, M. Probability and statistics for engineers and scientists, by r.e. walpole, r.h. myers, and s.l.

Science, T. D. (2018). The random forest algorithm.