Instacart Market Basket Analysis

Big Data Management and Analytics CS 6350.0U1 Final Project Report

Indrajit Gohokar, Piyush Supe, Raghav Mathur, Melvin James,
Erik Jonsson School of Engineering & Computer Science
The University of Texas at Dallas

INTRODUCTION

Instacart Market Basket Analysis is a part of the Kaggle community aiming to use Instacart's data of customer orders over time and predict which previously purchased products will be in a user's next order. Market Basket Analysis is a modelling technique which measures purchasing pattern for customers and is based on the fact that a consumer is more likely to buy a product which was present in his previous orders. In Instacart's competition, we are to forecast the products whether a user will buy again if he has already bought a certain product. It is a very interesting technique which has multiple uses in every domain. For example, to recommend goods for online e-commerce websites using data driven approach for easy and helpful shopping experience. In our project, we have identified this as a regression and classification problem and predicted products for given test dataset orders by not only using existing features but also deriving more features after understanding and thoroughly analysing the dataset.

The aim for this project is to be able to predict with accuracy the product-id of items for all the order-ids present in the test dataset and help provide an easy and decisive method for the task. We have implemented techniques like RandomForest, GradientBoostedTrees and DecisionTree regression to base our conclusion. Our project is part of an active competition by Kaggle. There are previous works done by skilled Data Scientists in this field, our work here will add to the pool of learners that have been already identified for the task.

PROBLEM DEFINITION AND ALGORITHM

i. Problem Definition

We are facing the problem of predicting products for a user where we have a multitude of features in the dataset. This is both an interesting and challenging problem. We are given many datasets for orders, products and user information consisting of features including: 'orderID', 'productID', 'add_to_cart_order', 'reordered', 'userID', 'order_number', 'order_dow', 'order_hour_of_day', 'days_since_prior_order' distributed across these datasets. We had to analyse these huge datasets and identify the attributes and build a classification and regression model which was efficient both in terms of accuracy and time complexity and can predict the products that will be placed in future orders. Apart from the given attributes, we also derived more attributes which helped us to improve our predictions.

The final training dataset created after joining the concerned datasets and pre-processing contains around 73008 products and 17 features for 10,000 (9367 - prior, 407 - train, 226- test) orders due to memory constraints. When dataset has so many entries, it takes a very long time for the processor to create a classification model. Also, it took a significant amount of time to join the datasets. We wrote classifiers in Python using PySpark and Ml-lib libraries on Databricks cluster.

ii. Algorithm Definition

We are using Supervised Learning Techniques to predict the products. For classification of products into their respective orders, we assumed a threshold value, for example= 0.19. In case the threshold value for that productID, orderID is greater than 0.19, we add that product in the respective order during prediction

Packages used in our project:

- Python:
  ```
  import numpy as np
  import pandas as pd
  ```
- Spark:
  ```
  from pyspark.sql import SparkSession
  from pyspark import SparkContext
  ```
- Machine Learning:
  ```
  from pyspark.mllib.regression import LabeledPoint
  from pyspark.mllib.tree import GradientBoostedTrees, GradientBoostedTreesModel
  from pyspark.mllib.tree import RandomForest, RandomForestModel
  from pyspark.mllib.tree import DecisionTree, DecisionTreeModel
  from pyspark.mllib.util import MLUtils
  from pyspark.ml.linalg import Vectors
  from pyspark.mllib.evaluation import MulticlassMetrics
  ```

DATASET DESCRIPTION

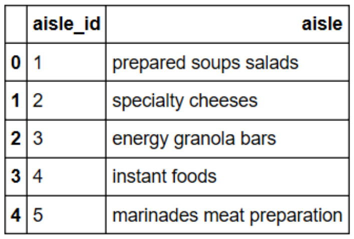
i. Source

The dataset is from Instacart.com. It is a part of Instacart Market Basket Analysis Kaggle Competition [https://www.kaggle.com/c/instacart-market-basket-analysis] which has a relational set of files describing customers' orders over time.

ii. Tables

The dataset for the Instacart Market Basket Analysis challenge includes 6 files namely the ‘product aisles’, ‘product departments’, the ‘products’ itself, ‘orders’, ‘order_products_prior’ and ‘order_products_train’.

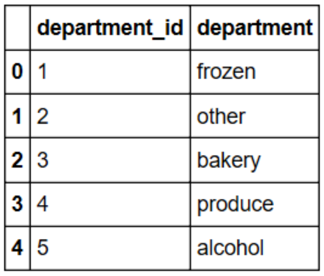
1. Aisles



	aisle_id	aisle
0	1	prepared soups salads
1	2	specialty cheeses
2	3	energy granola bars
3	4	instant foods
4	5	marinades meat preparation

Table 1 Aisles

2. Departments



	department_id	department
0	1	frozen
1	2	other
2	3	bakery
3	4	produce
4	5	alcohol

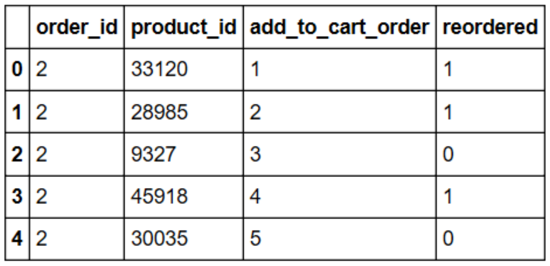
Table 2 Departments

3. Products

productID	product_name	aisle_id	department_id
1	Chocolate Sandwich Cookies	61	19
2	All-Seasons Salt	104	13
3	Robust Golden Unsweetened Oolong Tea	94	7
4	Smart Ones Classic Favorites Mini Rigatoni With Vodka Cream Sauce	38	1
5	Green Chile Anytime Sauce	5	13
6	Dry Nose Oil	11	11

Table 3 Products

4. Order_products_prior



	order_id	product_id	add_to_cart_order	reordered
0	2	33120	1	1
1	2	28985	2	1
2	2	9327	3	0
3	2	45918	4	1
4	2	30035	5	0

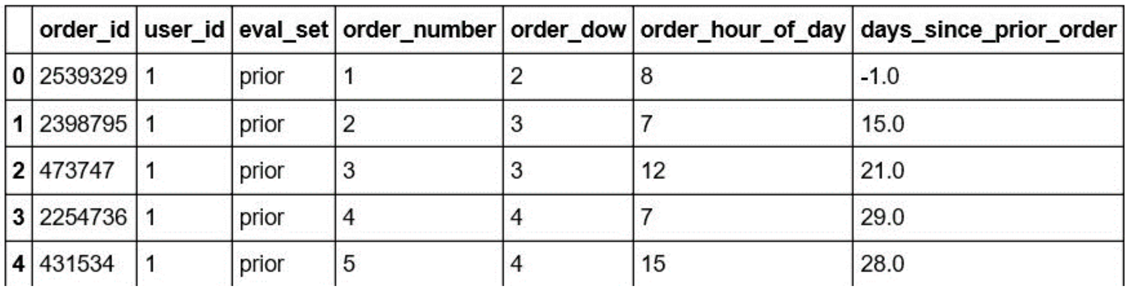
Table 4 Prior Orders

5. Order_products_train

orderID	productID	add_to_cart_order	reordered
1	49302	1	1
1	11109	2	1
1	10246	3	0
1	49683	4	0
1	47209	7	0
1	22035	8	1
36	39612	1	0
36	19660	2	1

Table 5 Train Orders

6.Orders



	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order
0	2539329	1	prior	1	2	8	-1.0
1	2398795	1	prior	2	3	7	15.0
2	473747	1	prior	3	3	12	21.0
3	2254736	1	prior	4	4	7	29.0
4	431534	1	prior	5	4	15	28.0

Table 6 All Orders

Additional dataset specific to training the model:

- Number of Final Training Instances- 24773
- Number of Test Instances- 14713
- Number of Features- 17
- Number of Classes- 2 (0, 1)
- Attribute-Information- ‘user_SumOrders', 'user_items_purchased_total', ‘distinct_items', 'user_avg_orderingGap', 'user_average_items', 'order_dow', 'order_hod', 'days_since', 'days_sinceProportion', 'numorders', 'numreorders', 'prod_reorderProportion', 'userProduct_orders', 'userProduct_ordersRatio', 'userProduct_average_cartOrder', 'userProduct_reorderProportion', 'userProduct_orders_sinceFinal'

In the final dataset, we used the following:

- 9367 prior orders
- 407 train orders
- 226 test orders

iii. Datasets

Test Dataset: The test dataset has same specification as the train data except for the classification column in the dataset. Also, the test data provided does not have the label column as it was in the training dataset.

Train Dataset: The Train data is composed of many derived features like- 'user_SumOrders','user_items_purchased_total',‘distinct_items','user_avg_orderingGap','user_average_items','order_dow','order_hod','days_since','days_sinceProportion','numorders','numreorders','prod_reorder_ratio','userProduct_orders','userProduct_ordersRatio','userProduct_average_cartOrder', 'userProductReorderRate' ,'userProduct_orders_sinceFinal'. We take the Order-Id and Product-Id as a key and output label as 0 or 1. If a product is predicted to be in that order, it will have label =1 else label =0.

Output Requirements: The ‘Kaggle’ competition requires us to submit the prediction of the test order-id’s in the form of Order-Id and List of Product-Ids in a file. They will in return provide us with the accuracy. Since it was not possible to use the complete dataset on Databricks assuming the Cluster size constraints, we made predictions on using 10000 orders. Hence, we evaluated our models by sampling the train dataset and calculating accuracy using Evaluation Metrics.

iv. Preprocessing the dataset

The entire dataset of the orders given for the market basket analysis challenge consisted of over 3.4 million orders. We used 10,000 (9367 priors, 407 train and 226 test) orders for training the regression models. Since the dataset was already in the required format we were not required to perform extensive pre-processing technique required except for a few steps.

In the orders dataset, for the first order of every user the ‘days_since_prior_order’ field was null as there are no orders before that for the user. So, for every such order, NULL was replaced with 0 in the dataset. The ‘eval_set’ column in the orders dataset had categorical labels for the type of order it was, ‘prior’ if it was from the prior order dataset, ‘train’ if it was from train order dataset and ‘test’ if it was from the test dataset. This was changed to numerical categories - 0 for the prior orders, 1 for the train orders and 2 for the test orders.

In the later part of the project, we believed to JOIN multiple tables to obtain user, order and products related information and finally run our classification models on the same.

RELATED WORK

Since it a new competition, it has around 2382 submission on Kaggle.com with the highest score of 0.4084459 F-Score. A lot of work is being done this using Gradient Boosting, Regression, and Random Forest techniques since it is prominent area of research and development and has its application in various fields. Our work will add to this for future enhancements and research.

EXPERIMENTAL EVALUATION AND RESULTS

i. Methodology

Our proposed solution is based on predicting whether a product from the customer's previous order would be present in the test order or not. We have performed our experiments using Python as the programming languages for RDD Creation, joining datasets and for implementing Machine learning techniques. The decision to choose appropriate classifiers was based on various factors like Size of dataset, Dimensionality of Feature Space, Overfitting, and Efficiency in terms of Speed, Performance, and Memory Usage. We have created RDDs of datasets for achieving higher performance. We realized that the features are not sufficient to make predictions for test orders. Hence, we derived other attributes using existing features. We added the following features to the dataset-

- 'user_SumOrders': Total orders placed by a user
- 'user_items_purchased_total': Total items purchased by a user
- 'distinct_items': Distinct items bought by a user over all his orders
- 'user_avg_orderingGap': Average gap in days between a user's orders
- 'user_average_items': Average no. of items bought by a user in very order
- 'order_dow': Day of Week of an order
- 'order_hod': Hour of Day for an order
- 'days_since': Days since previous order
- 'days_sinceProportion': It is equal to 'days_since' divided by 'user_avg_orderingGap'
- 'numorders': No of times a product appeared in any order
- 'numreorders': No of times a product was reordered by any user
- 'prod_reorderProportion': It is equal to 'numreorders' divided by numorders'
- 'userProduct_orders': Total orders placed by a user for a product
- userProduct_ordersRatio': It is equal to 'userProduct_orders'/'user_SumOrders'
- 'userProduct_average_cartOrder': It is equal to 'sum_cartOrder'/'userProduct_orders'.
- 'userProduct_reorderProportion': It is equal to 'userProduct_orders' / 'user_SumOrders'
- 'userProduct_orders_sinceFinal': It is equal to 'user_SumOrders'- 'userProduct_final_orderId'

We used regression techniques to obtain threshold values keeping (orderId, productid) as the key. Based on the threshold level we finally classified the products into their respective orders. After training the models and getting predictions, we evaluated performance using metrics such as Precision, F1 Score and Recall. Since we did not have actual test labels, we had to split the training dataset randomly into training and test partitions using sample function and evaluate our models.

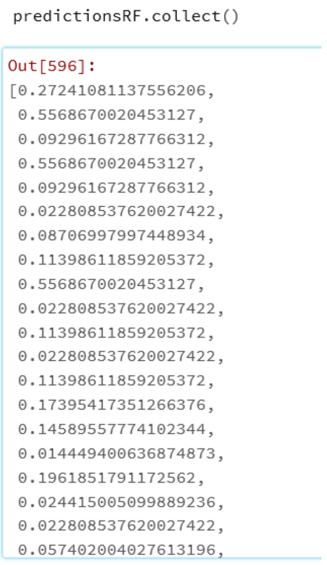
Classification Performance Evaluation Metrics-

- **Cross Validation-** to know, how accurately a model will perform on test dataset
- **Precision and Recall-** are Positive Predicted value and Sensitivity respectively and are based on degree of relevance
- **F1 Score-** is a weighted average of Recall and Precision, which has its best value at 1 and worst at 0.

Regression Techniques used are as follows:

We used regression techniques to obtain threshold values keeping (orderId, productid) as the key.

a) RandomForest.trainRegressor

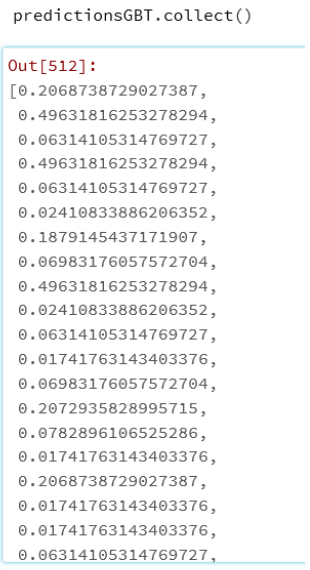


```
predictionsRF.collect()

Out[596]:
[0.27241081137556206,
 0.5568670020453127,
 0.09296167287766312,
 0.5568670020453127,
 0.09296167287766312,
 0.022808537620027422,
 0.08706997997448934,
 0.11398611859205372,
 0.5568670020453127,
 0.022808537620027422,
 0.11398611859205372,
 0.022808537620027422,
 0.11398611859205372,
 0.17395417351266376,
 0.14589557774102344,
 0.014449400636874873,
 0.1961851791172562,
 0.024415005099889236,
 0.022808537620027422,
 0.057402004027613196,
```

Figure 1 Results for Regression Techniques:
Predicted threshold values of each ProductID with respect to OrderID

Based on the results obtained from the Regression models and using a threshold value of 0.19 we decided whether to include the product in the customers next order or not.

b) GradientBoostedTrees.trainRegressor

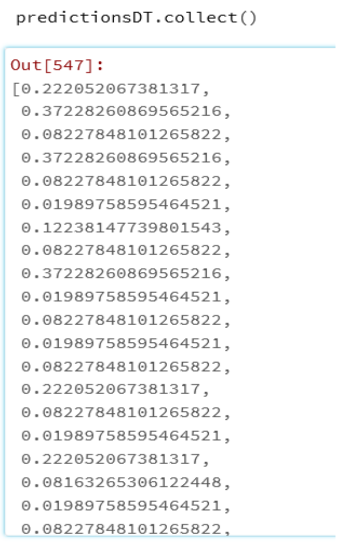


```
predictionsGBT.collect()

Out[512]:
[0.2068738729027387,
 0.49631816253278294,
 0.06314105314769727,
 0.49631816253278294,
 0.06314105314769727,
 0.02410833886206352,
 0.1879145437171907,
 0.06983176057572704,
 0.49631816253278294,
 0.02410833886206352,
 0.06314105314769727,
 0.01741763143403376,
 0.06983176057572704,
 0.2072935828995715,
 0.0782896106525286,
 0.01741763143403376,
 0.2068738729027387,
 0.01741763143403376,
 0.01741763143403376,
 0.06314105314769727,
```

Figure 2 Results for Regression Techniques:
Predicted threshold values of each ProductID with respect to OrderID

Based on the results obtained from the Regression models and using a threshold value of 0.19 we decided whether to include the product in the customers next order or not.

c) **DecisionTree.trainRegressor**



```
predictionsDT.collect()
```

```
Out[547]:
[0.222052067381317,
 0.37228260869565216,
 0.08227848101265822,
 0.37228260869565216,
 0.08227848101265822,
 0.01989758595464521,
 0.12238147739801543,
 0.08227848101265822,
 0.37228260869565216,
 0.01989758595464521,
 0.08227848101265822,
 0.01989758595464521,
 0.08227848101265822,
 0.222052067381317,
 0.08227848101265822,
 0.01989758595464521,
 0.222052067381317,
 0.08163265306122448,
 0.01989758595464521,
 0.08227848101265822,
```

Figure 3 Results for Regression Techniques:
Predicted threshold values of each ProductID with respect to OrderID

Based on the results obtained from the Regression models and using a threshold value of 0.19 we decided whether to include the product in the customers next order or not.

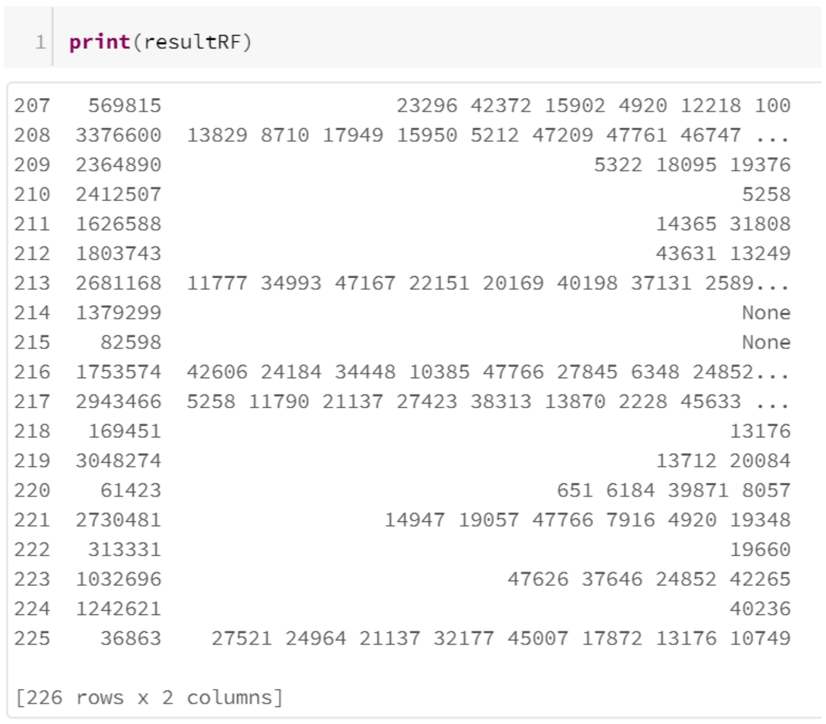
Classification Techniques used are as follow:

a) **Random Forest-**

They are Ensemble learning machine learning techniques made up of many decision trees each of which is trained using a bootstrap sample of the training data by randomly selecting a subset of features. By creating a model, its classification is based on Mean prediction of all the trees. We used a Random Forest model on our Instacart's anonymized dataset. We tuned out model by changing its parameters like No. of Trees. We found the best set of values of the parameters numTrees = 3. The performance or accuracy of the model was calculated using Precision, Recall and F1 measure and Cross Validation. The accuracy on the training dataset comes out to be approx. 88.9%.

Outputs:

All the product list was created based on the results of the previous step. If there are no products in that order, the product list is replaced by 'None'.



```python
print(resultRF)
```

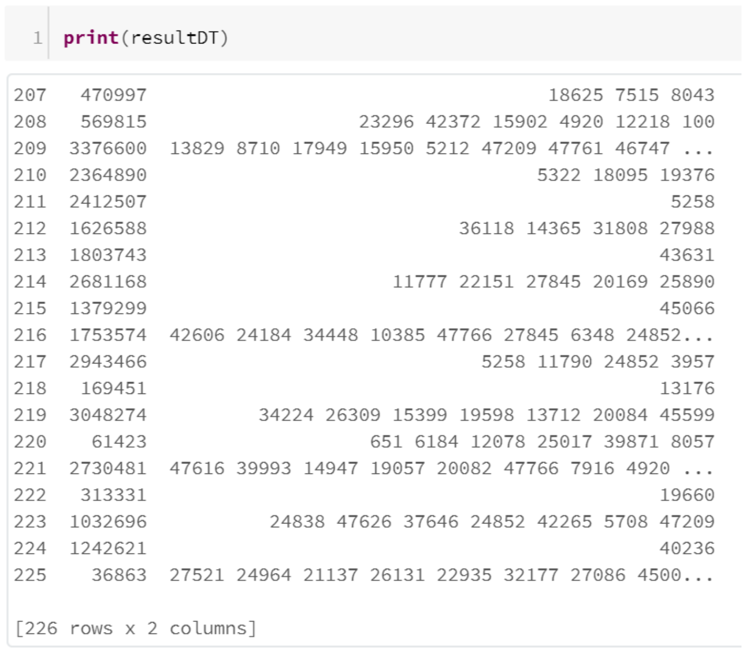
```
207   569815                   23296 42372 15902 4920 12218 100
208  3376600  13829 8710 17949 15950 5212 47209 47761 46747 ...
209  2364890                                  5322 18095 19376
210  2412507                                              5258
211  1626588                                       14365 31808
212  1803743                                       43631 13249
213  2681168  11777 34993 47167 22151 20169 40198 37131 2589...
214  1379299                                              None
215    82598                                              None
216  1753574  42606 24184 34448 10385 47766 27845 6348 24852...
217  2943466  5258 11790 21137 27423 38313 13870 2228 45633 ...
218   169451                                             13176
219  3048274                                       13712 20084
220    61423                                651 6184 39871 8057
221  2730481               14947 19057 47766 7916 4920 19348
222   313331                                             19660
223  1032696                           47626 37646 24852 42265
224  1242621                                             40236
225    36863    27521 24964 21137 32177 45007 17872 13176 10749

[226 rows x 2 columns]
```

Figure 4 Result of Random Forest Classifier

b) **Decision Trees-** is a predictive modelling technique that proceeds by looking at observations about an item which is denoted in terms of branches. It uses Information gain of each attribute to classify an instance, finally reaching the target values (or classes) denoted by leaf nodes of the tree. In the input parameters were chosen to be impurity='variance', maxDepth=5, maxBins=32. The accuracy on the training dataset comes out to be approx. 89.1%.

Output:

All the product list was created based on the results of the previous step. If there are no products in that order, the product list is replaced by ‘None’.



```python
print(resultDT)
```

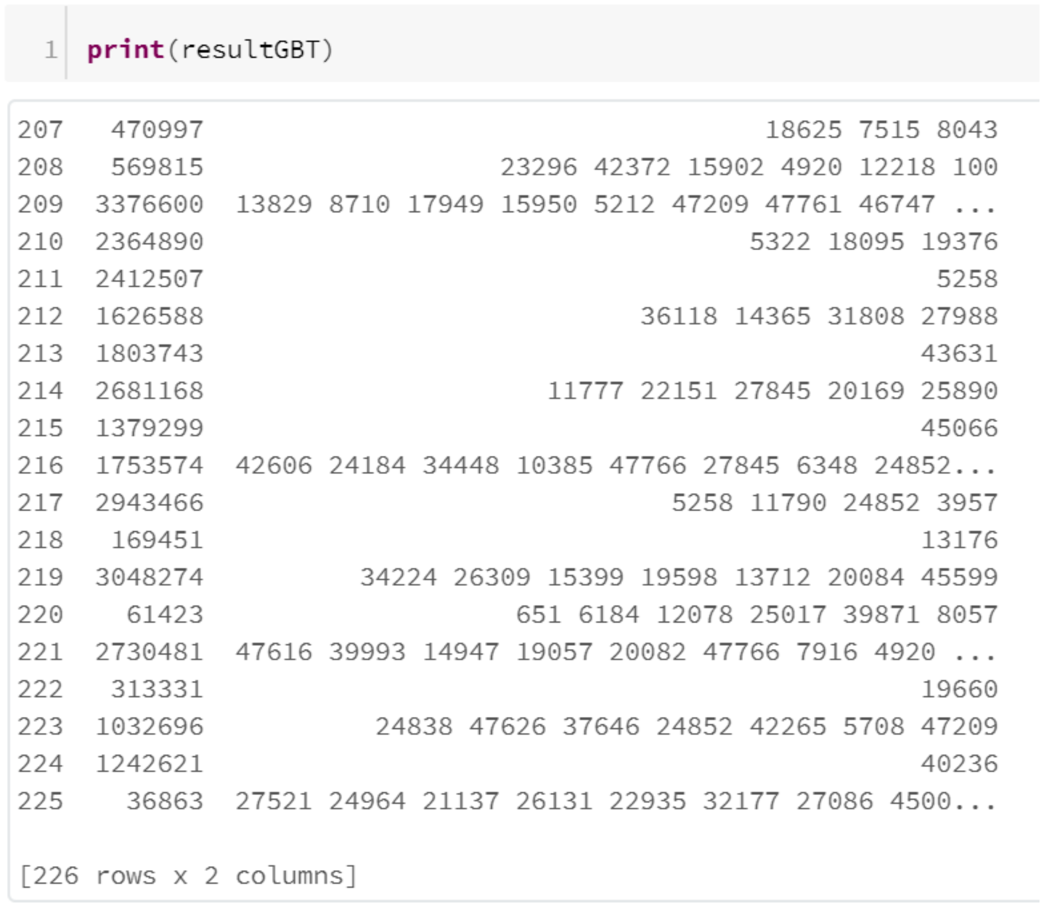
```
207   470997                                   18625 7515 8043
208   569815                   23296 42372 15902 4920 12218 100
209  3376600  13829 8710 17949 15950 5212 47209 47761 46747 ...
210  2364890                                  5322 18095 19376
211  2412507                                              5258
212  1626588                           36118 14365 31808 27988
213  1803743                                             43631
214  2681168                     11777 22151 27845 20169 25890
215  1379299                                             45066
216  1753574  42606 24184 34448 10385 47766 27845 6348 24852...
217  2943466                             5258 11790 24852 3957
218   169451                                             13176
219  3048274          34224 26309 15399 19598 13712 20084 45599
220    61423                    651 6184 12078 25017 39871 8057
221  2730481  47616 39993 14947 19057 20082 47766 7916 4920 ...
222   313331                                             19660
223  1032696           24838 47626 37646 24852 42265 5708 47209
224  1242621                                             40236
225    36863  27521 24964 21137 26131 22935 32177 27086 4500...

[226 rows x 2 columns]
```

Figure 5 Result of Decision Tree Classifier

c) **Gradient Boosted Trees-** are quite popular techniques for classification and regression as they use ensembles of weak prediction models like decision trees. It uses a more regularized and advanced interface of model validation to regulate over-fitting, hence has better performance. It can be run on a cluster and allows for parallel computation on a single machine. The parameters were chosen to be ‘numIterations’ =3. From our experiments, we concluded that as we increase the no. of iterations, we get a higher accuracy. We obtained an accuracy of 88.9 % on the training dataset.

Output:

All the product list was created based on the results of the previous step. If there are no products in that order, the product list is replaced by ‘None’.



```
print(resultGBT)
```

```
207   470997                                           18625 7515 8043
208   569815                          23296 42372 15902 4920 12218 100
209  3376600  13829 8710 17949 15950 5212 47209 47761 46747 ...
210  2364890                                          5322 18095 19376
211  2412507                                                      5258
212  1626588                                36118 14365 31808 27988
213  1803743                                                     43631
214  2681168                          11777 22151 27845 20169 25890
215  1379299                                                     45066
216  1753574  42606 24184 34448 10385 47766 27845 6348 24852...
217  2943466                                    5258 11790 24852 3957
218   169451                                                     13176
219  3048274            34224 26309 15399 19598 13712 20084 45599
220    61423                        651 6184 12078 25017 39871 8057
221  2730481  47616 39993 14947 19057 20082 47766 7916 4920 ...
222   313331                                                     19660
223  1032696             24838 47626 37646 24852 42265 5708 47209
224  1242621                                                     40236
225    36863  27521 24964 21137 26131 22935 32177 27086 4500...

[226 rows x 2 columns]
```

Figure 6 Result of Gradient Boosted Trees Classifier

CONCLUSION

Predicting the purchasing pattern of users is of significance use to E-commerce application. It provides tremendous unseen benefits like- Better Consumer Understanding, Cognitive Purchasing, Increasing Revenues of Companies, Improving Sales and assisting buyers while shopping. In our project, we followed the classification approach to solve this problem which makes it easy to predict items in a user's future order. We wished to use the complete dataset of 3.2 million instances but could not due to memory constraints. Still, we could achieve a competitive accuracy and performance by using Spark. We anticipated that if we had more accurate features in our dataset, we might obtain better results in classification. So, we derived multitude of attributes after deep analysis of dataset and problem domain. We could not use pipelining as we had to perform feature generation and because of the complexity of features, it did not pertain to our dataset. Using PySpark, we could join large datasets, make DataFrames and build Models within seconds and get predictions in couple of minutes. It took around 20mins to get DataFrames. Also, we achieved a challenging accuracy over the performance metrics. In our case, we achieved the best performance using Decision tree classifier on our dataset, though it was a marginal win. We believe, since we had only 2 categorical variables (0,1), Decision Tree could perform the best.

Classifier	**Precision**	**Recall**	**F1-Score**
Random Forest	0.889	0.889	0.889
Gradient Boosting	0.889	0.889	0.889
Decision Tree	0.891	0.891	0.891

Table 7 Comparison of Experimental Results

CONTRIBUTIONS

Everyone contributed equally to the project while working either on Python/Spark/Scala. We all worked on extensively to analyze the dataset and feature creation and extraction which was a major part of the project. Based on execution times and accuracy we chose to use Databricks cluster for our project.

- Dataset Discovery and Preprocessing: Indrajit Gohakar, Raghav Mathur
- Project Status report (First phase): Piyush Supe, Raghav Mathur, Melvin James
- Programming contribution-
  - Indrajit Gohakar: Worked on Data Joins and RDD creations
  - Raghav Mathur: Worked on Preprocessing and Random Forest technique
  - Melvin James: Worked on data analysis and Decision Trees technique
  - Piyush Supe: Worked on Gradient Boosted Trees technique
- Logging of experimental results and deriving conclusions: Melvin James, Piyush Supe
- Reporting and analysis: Raghav Mathur, Indrajit Gohakar, Piyush Supe

REFERENCES

I. http://www.albionresearch.com/data_mining/market_basket.php

II. https://www.kaggle.com/c/instacart-market-basket-analysis

III. https://www.kaggle.com/c/instacart-market-basket-analysis/kernels

IV. http://spark.apache.org/docs/latest/ml-classification-regression.html

V. https://spark.apache.org/docs/latest/mllib-ensembles.html

VI. https://weiminwang.blog/2016/06/09/pyspark-tutorial-building-a-random-forest-binary-classifier-on-unbalanced-dataset/

VII. http://spark.apache.org/docs/2.1.0/api/python/pyspark.sql.html

VIII. https://select-statistics.co.uk/blog/market-basket-analysis-understanding-customer-behaviour/

IX. https://en.wikipedia.org/wiki/Association_rule_learning

X. https://webfocusinfocenter.informationbuilders.com/wfappent/TLs/TL_rstat/source/topic43.htm