Customer segmentation based on E-commerce transactions

Lorenzo Simone[1], **Marco Sorrenti** [2], **Simone Baccile** [3]

[1] Università di Pisa (Artificial Intelligence); l.simone3@studenti.unipi.it
[2] Università di Pisa (Data and Knowledge: Science and Technologies); m.sorrenti4@studenti.unipi.it
[3] Università di Pisa (Data and Knowledge: Science and Technologies); s.baccile@studenti.unipi.it

Version January 3, 2021 submitted to Data Mining

1. Data Understanding

The dataset we are analyzing, which name is `customer_supermarket.csv`, contains informations about the historical sales of a supermarket company which have been recorded for approximately one year, from *1-12-10* to *9-12-11*. The main purpose of this section is to understand which features could affect and better describe the customer behavior, in order to formalize a customer profile as long with his purchasing behavior. This dataset contains 471910 observations of 9 variables. We describe a tabular semantic clarification for each variable in Table 1.

Table 1. Description of the semantic clarification of each column for the analyzed dataset

Column	Description	Category	Type
CartID ← ~~*BasketID*~~	ID of transaction with unique customer and date time	Categorical	Object
CartDate ← ~~*BasketDate*~~	Purchase date time in timestamp format	Categorical	Object
UnitPrice ← ~~*Sale*~~	Price of a single item, unique by *ProductID*	Numerical	Float64
CustomerID	ID representing each different customer	Categorical	Object
CustomerCountry	Birth country of the purchasing customer	Categorical	Object
ProductID ← ~~*ProdID*~~	ID representing each different product	Categorical	Object
ProductDescription ← ~~*ProdDescr*~~	Description of the associated *ProductID*	Categorical	Object
Quantity ← ~~*Qta*~~	Number of purchased items for each transaction	Numerical	Int64

As a preliminary step we edited each column identifier having an ambigous name by a more expressive and a more self-explaining one.

By investigating values for each transaction, we noticed the presence of missing values, that will need a dedicate procedure in the data cleaning phase:

- **CustomerID**: 65080 missing values.
- **ProdDescr**: 753 missing values.

1.1. Feature analysis

Starting from the informations that we have on each feature from the dataset, we focus on analyzing aspects of the numerical ones, which could pave the way for a better future data cleaning phase. In Table 2 we report the Inter-Quartile range (IQR) calculation for the price of a single product and the purchased quantity in each transaction.

Table 2. Inter-Quartile Range calculation (IQR) for numerical discrete features (UnitPrice, Quantity)

Feature	IQR
UnitPrice	2.5
Quantity	11

In Figure 1 we report the associated boxplot visualization, it is clearly noticeable how far certain observations are distributed from the others, highlighting the need of an outlier removal procedure.

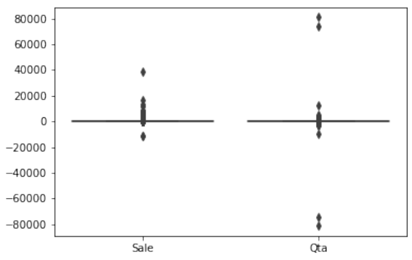


Figure 1. Boxplot visual representation of numerical discrete features before the outlier detection and removal procedures.

1.2. Data cleaning

In this section, we list some of the most important operations that we have done to clean data from outliers, missing values, and other strange situations that we encountered in the exploration of the dataset. The Table 3 lists what kind of operations we have done on which columns, with respective reason and the number of rows that have been modified by our decisions.

Table 3. Cleaning operations performed on the dataset

Column	Operation	How	Reason	Number of rows
Qta	DROP	$Qta \geq 3500 \land Qta \leq 0$	We have chosen the threshold of elements to drop using inter-quartile distance and the standard deviation over the values of the Qta column	9758
Sale	DROP	$Sale \geq 200 \land Sale \leq 0$	We have chosen the threshold of elements to drop using inter-quartile distance and the standard deviation over the values of the Sale column	826
CustomerID	UPDATE	$CustomerID = null$	Since we have a lot of null CustomerID, we decided to create a custom CustomerID concatenating the letter 'G' with relative BasketID where CustomerID is null	63518 (1268 new) CustomerID)
ProductID	DROP	Useless ProductID	We found specific code for some ProductID that did not represent real products	1693

In the Figure 2 is represented the distribution of rows, by Quantity and Sale, after the cleaning.

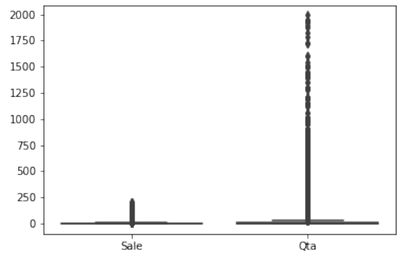


Figure 2. Boxplot visual representation of numerical discrete features after data cleaning process.

In the Table 4 there is list of all particular ProductID that we have dropped.

Table 4. Description of ProductID dropped during data cleaning operations

ProductID	Description
S	Sample products
B	Debts
M,m	Manual corrections of CartID cost
D	Discounts
C2	Carriage fee
CRUK, POST	Shipping fees
AMAZONFEE	Amazon fee
BANK CHARGES	Recharge of the account

At the end of this step, we reduced the size of the dataset from 471910 rows to 459612 and we managed all missing values in the dataset.

1.3. Feature Engineering

In this phase, we found new useful features from the dataset, that we can exploit to extract more information and knowledge about the behavior of customers.

Table 5. Description of useful features extracted from dataset

New feature	Type	Description
CartYear	int64	Year of purchase
CartMonth	int64	Month of purchase
CartDay	int64	Day of purchase
CartHour	float64	Hour of purchase
Holiday	bool	Boolean that is true if day of purchase is an holiday
Sale	float64	$UnitPrice * Quantity$
TotalCartPrice	float64	Total cost of cart purchased
TotalCartItems	int64	Number of items purchased in one cart
ProductType	object	Type of product that can be classified as *cheap* or *average* or *expensive*

In Table 5 to assign the value of ProductType column we tried to balance the distribution of products. We have for **cheap** products $0 < UnitPrice \leq 2$, for **average** products $2 < UnitPrice < 4$ and for **expensive** products $UnitPrice \geq 4$.

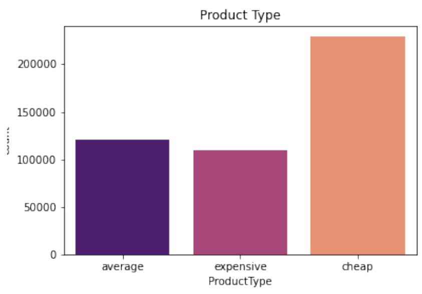


Figure 3. Distribution of ProductType feature

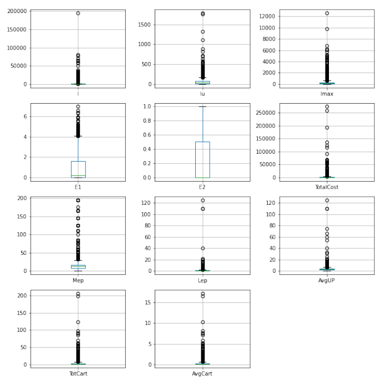
1.4. Data Preparation

For this task, we have constructed a new dataset containing only interesting features that we will use in subsequent tasks. First of all, we extract all unique CustomerID in the dataset and put them in the new dataset. Now we assign to each customer the most representative Country, which is the Country where the customer has made the most purchases. Then we extract new features as described in Table 6.

Table 6. List of mandatory and new features extracted for studying the customer behavior

Feature	Description
I	Total number of items purchased by a customer
Iu	Number of distinct items bought by a customer
Imax	The maximum number of items purchased by a customer during a shopping session
E1	Shannon entropy of the number of product of each basket related with the total number of product purchased by each customer
E2	Shannon entropy of holiday baskets of each customer
TotalCost	Total cost of all basket purchased by a customer
Mep	Most expensive product bought by a customer
Lep	Less expensive product bought by a customer
AvgUP	Average UnitPrice of all products bought by a customer
Totcart	Total number of basket purchased by each customer
Totcart	Total number of basket purchased by each customer
AvgCart	Average monthly basket bought by a customer

After the extraction of these features, we did another cleaning phase on the new customer dataset to eliminate outlier values. We did it with the same methodology that we used in the previous step, using the inter-quartile range, standard deviation, and box-plot, as shown in Figure 4.


(a)

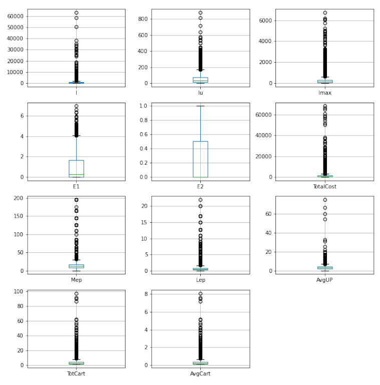

(b)

Figure 4. (**a**) Boxplots before the cleaning; (**b**) Boxplots after the cleaning

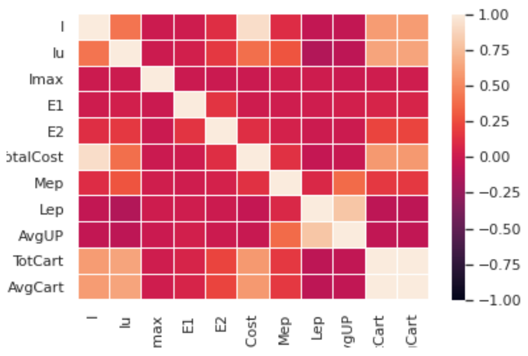
2. Clustering Analysis

2.1. K-Means clustering

In order to prevent that different features with larger scales value are dominant with respect to others we perform a standardization following the rule:

$$x_i \simeq \frac{(x_i - \mu)}{\sigma} \tag{1}$$

We then proceed by investigating for interesting properties amongst them such as dependency and feature correlation. We computed a correlation matrix $C = \frac{A^T \times A}{(n-1)}$ from $A^{n \times m}$ representing the dataset matrix of n instances of m features. In Figure 5 we propose an heatmap visualization of C, a symmetric correlation matrix, showing a full positive dependance on the diagonal and more meaningful insights between other features such as positive or negative correlation.



(a)

Figure 5. Heatmap plot of C correlation matrix between features

With the objective of dimensionality reduction of the feature space by spotting redundant patterns among features, we performed PCA analysis using SVD of the matrix A. The result of the decomposition of matrix A is

$$A = USV^T, \tag{2}$$

where U is a unitary matrix and S is the diagonal matrix of singular values s_i. Given this we could review the matrix C in a simpler definition

$$C = \frac{VSU^TUSV^T}{(n-1)} = V\frac{S^2}{(n-1)}V^T, \tag{3}$$

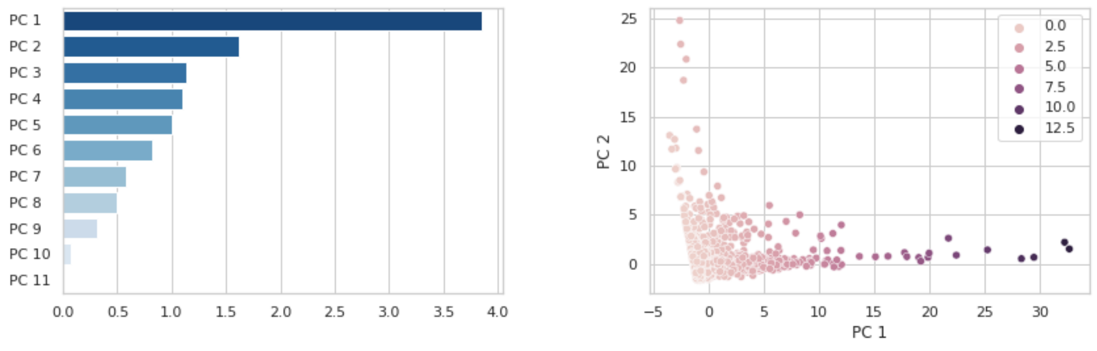


Figure 6. (**a**) Histogram plot fo the principal components variance $Var(PC_i)$; (**b**) First two principal component 2D visualization plot. Color gradient is weighted by $Var(PC_1)$, $Var(PC_1)$

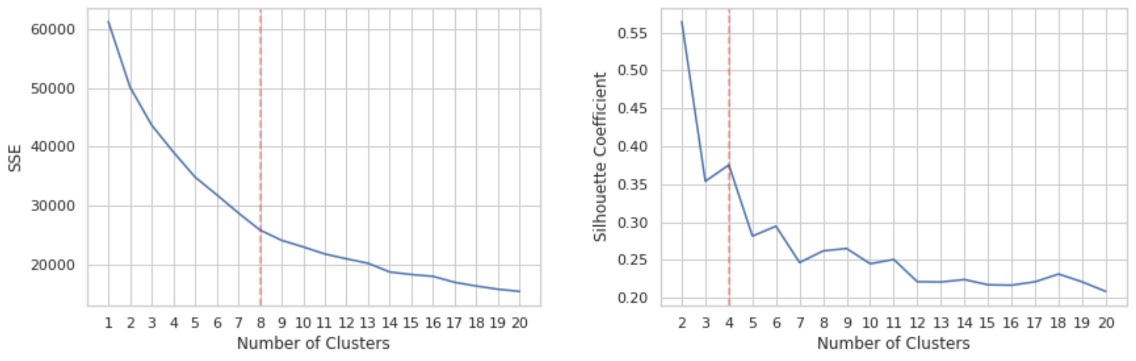
From now on we can conclude that right singular vectors V are principal directions and singular values are related to the eigenvalues of C given $\lambda = \frac{s_i^2}{(n-1)}$. In order to perform a k-rank approximation of the SVD which had the best trade-off in terms of data significance, we visualize an histogram plot of variances for each principal component vector given by $AV = USV^TV = US$ in Figure 6.a. In Figure 6.b we depicted a scatter plot of the first principal component with respect to the second one, as we can see it is difficult to spot different clusters utilizing only 2 dimensions.

2.1.1. *Optimal "K" centroids parameter*

The evaluation of the best value for the parameter "k", indicating the distinct number of centroids has been achieved by analyzing different runs of the K-means algorithm with different initializations ranging from 1 to 20 different clusters. For each of those runs we analyzed throughout different evaluations the Sum of Squared Error (SSE) and the Silhouette coefficient.
We performed an *elbow analysis* according to the SSE values, utilizing the formula:

$$SSE = \sum_{i=1}^{k} \sum_{x \in C_i} dist(m_i, x)^2 \tag{4}$$

calculating the distance to the nearest cluster C_i for each point m_i (Figure 7.a). For a better comprehension of the optimal value we also estimated how similar a point is to its own cluster compared to other clusters by the Silhouette coefficient (Figure 7.b).



(a)

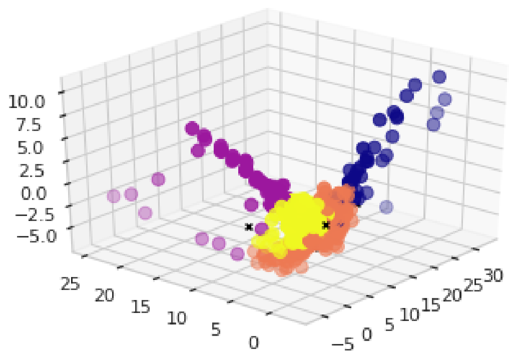
Figure 7. (**a**) Sum of Squared Error (SSE) over different runs of the K-means; (**b**) Silhouette coefficient over different runs of the K-means algorithm, an optimal value can be found along a red vertical dashed line (x=4);

From the function describing SSE over number of clusters, we used the *kneedle* algorithm finding 8 as a reasonable value suggested by this specific analysis [1].

Despite in most cases this analysis seems sufficient, we do not disregard the suggestion from the Silhouette function plot indicating an optimal number of clusters of 4, as indicated by its peaks. Another useful information for an optimal number for the parameter *k*, has been pointed out by the hierarchical clustering analysis summarized in Section 2.2. We have decided to take into consideration the results proposed by the visualization of different dendrograms, which were also suggesting 4 as an optimal value for the number of centroids.

2.1.2. Clustering results

After the preliminary analysis for the optimal number of clusters, ran with the endorsement of the hierarchical clustering phase, we performed the K-means algorithm considering 10 random initialization of the centroid location in the space and the Euclidean distance metric, since we are not considering categorical features. We propose, in Figure 15, a 3D space scatter plot visualization with the help of PCA of the labeled clusters specifying the position of the four different centroids represented by a black *x*.



(a)

Figure 8. 3D scatter plot of the four main labeled clusters proposed by running K-means algorithm and PCA analysis extracting the top three principal components.

The preliminary analysis was needed in order to obtain a result maximizing the silhouette coefficient and minimizing the SSE metric from a quantitative analysis, while from a qualitative perspective each cluster seem to be fairly separated from the others.

2.2. Hierarchical clustering

There are two basic approaches:

- **Agglomerative** clustering: which starts with the point as an individual cluster and, at each step, merge the closest pair of clusters;
- **Divisive** clustering: which starts with one big cluster and at each step split it until only singleton clusters remain.

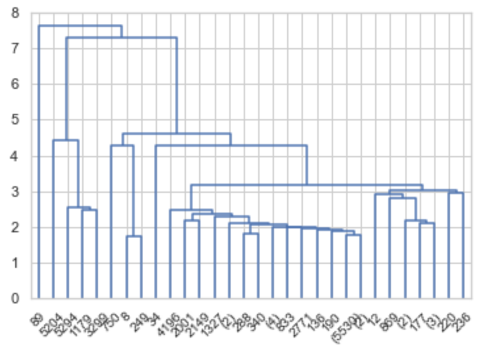
A hierarchical clustering is often displayed graphically using a tree-like diagram called **dendogram**. Since hierarchical clustering do not have to assume any particular number of clusters, it could be useful to cut the dendogram at the proper level.

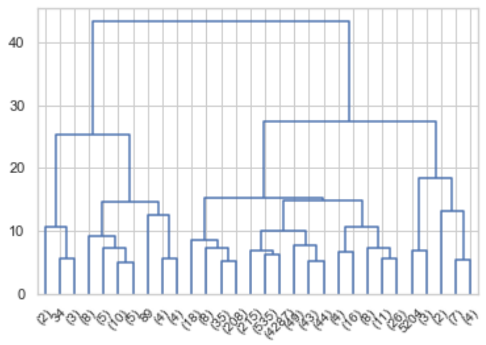
The key operation in the agglomerative clustering is the computation of the **proximity** of two clusters. According to the definition of cluster proximity, there are different techniques for the agglomerative hierarchical clustering:

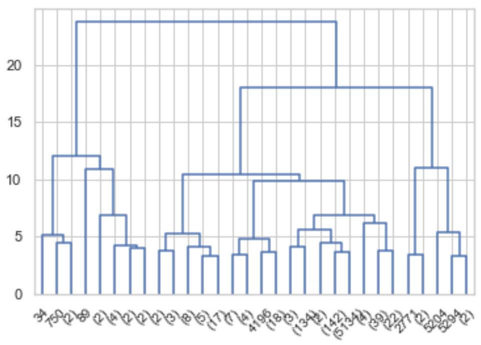
- **MIN**: for the *single link* version, the proximity of two clusters is defined as the minimum of the distance (maximum of similarity) between any two points in the two different clusters;
- **MAX**:for the *complete link* version, the proximity of two clusters is defined as the maximum of the distance (minimum of similarity) between any two points in the two different clusters;
- **GROUP AVERAGE**: for this version, the proximity of two clusters is defined as the average pairwise proximity among all pairs of points in the different clusters;
- **DISTANCE BETWEEN CENTROIDS**: for this version, the proximity of two clusters is defined by calculating the distance between the centroids of the two different clusters;
- **WARD'S METHOD**: for this version, the proximity of two clusters is defined as the increase in the squared error that results when two clusters are merged.

In order to reduce the number of variables we are working with, we decide to apply the PCA method, which main goal is to find a new set of features that better captures the variability of the data. Before applying PCA, we start from the dataset standardization, by StandardScaler, which standardize features by removing the mean and scaling to unit variance.

After that, we plot different dendrograms, according to the approach adopted in order to evaluate the proximity between clusters (distance).


(a)


(b)


(c)

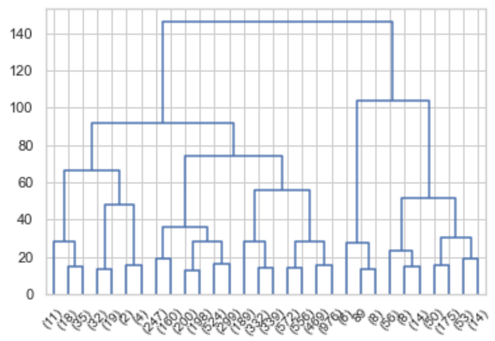

(d)

Figure 9. (**a**) Single Link or MIN version of the hierarchical clustering; (**b**) Complete Link or MAX version of the hierarchical clustering; (**c**) Group Average version of the hierarchical clustering; (**d**) Hierarchical clustering by Ward's method.

According to dendogram b (Complete Link or MAX version of the hierarchical clustering), we create an instance of **AgglomerativeClustering** using the euclidean distance as the measure of distance between points and complete linkage to calculate the proximity of clusters. As number of clusters we choose 4, by cutting the dendogram at the proper level (we set the parameter $n\_clusters = 4$).

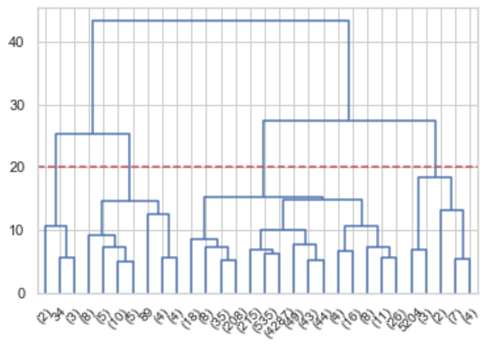
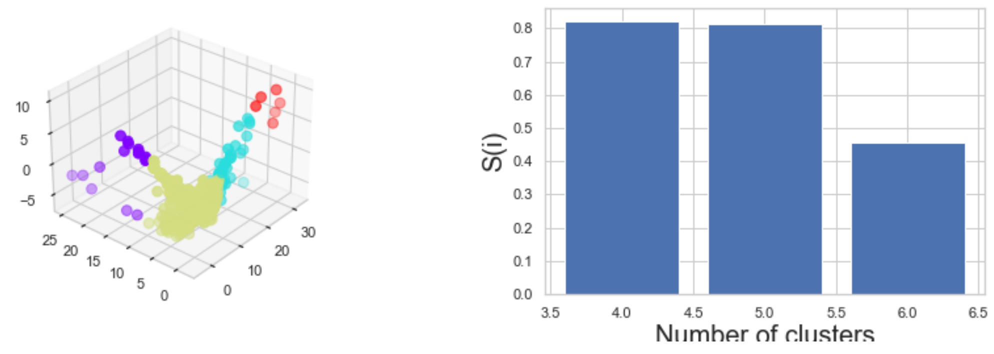

(a)

Figure 10. Dendogram built using the euclidean distance as the measure of distance between points and complete linkage to calculate the proximity of clusters.

Below, is possible to visualize the 4 clusters in 3D.



(a)

Figure 11. (**a**) 3D scatter plot of the four clusters proposed by running hierarchical clustering algorithm and PCA analysis extracting the top three principal components; (**b**) Comparing the silhouette score by running an instance of Agglomerative clustering based on Euclidean distance (4,5,6 number of clusters).

2.3. Density-Based Clustering

We have chosen DBSCAN algorithm to do density-based clustering. DBSCAN is a clustering method that groups points that are in density regions and mark outlier points in low-density regions. Mainly DBSCAN depends on two parameters:

- ϵ: specifying the radius of a neighborhood for some point.
- **minPts**: is the least number of points inside a radius ϵ, to classify a point as a *core point*.

First of all, in our analysis, we studied how to select the best values for ϵ and minPts. In the first case, we have chosen ϵ using the Nearest Neighbors algorithm and by the plot of the distances, we obtained the range of value where there is the knee of the curve [2]. Then we have compared it with the computation of Silhouette and Separation score over the DBSCAN algorithm with a fixed minPts. In this way we found that the best value is the one obtained by NN. We can see these evaluation in Figure 12.

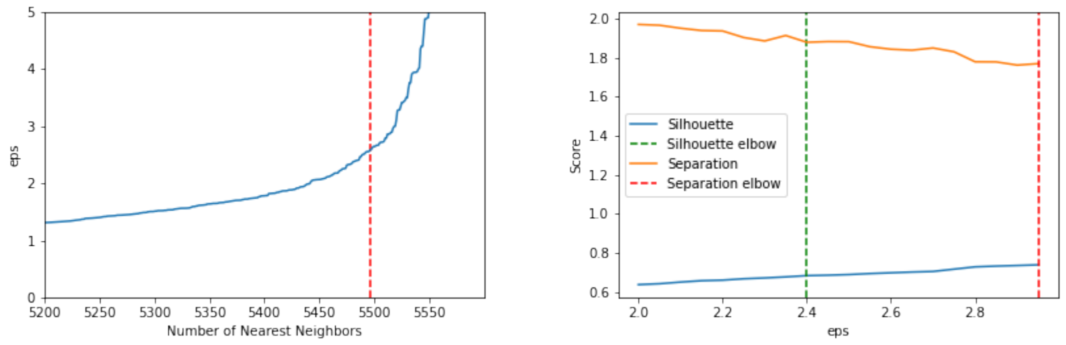


Figure 12. Analysis for the selection of a proper value for ϵ

Instead, for the selection of minPts on a multi-dimensional dataset, we performed multiple DBSCAN with fixed best ϵ to find the best minPts based again on Silhouette and Separation score. We can see evaluation of minPts in Figure 13.

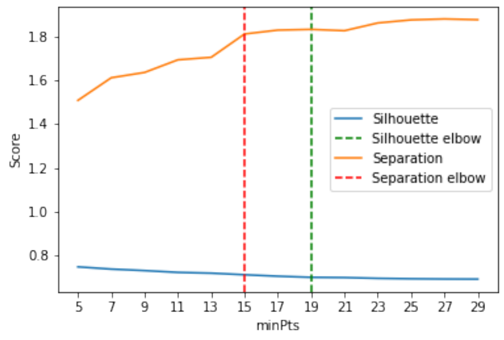


Figure 13. Analysis for the selection of a proper value for minPts

As a consequence of these evaluation, we defined $\epsilon \simeq 2.58$ and $minPts = 17$. Then we applied DBSCAN algorithm using these parameters and as a distance metric, the euclidean distance. In Figure 14 we can see the distribution of cluster over the mandatory features.

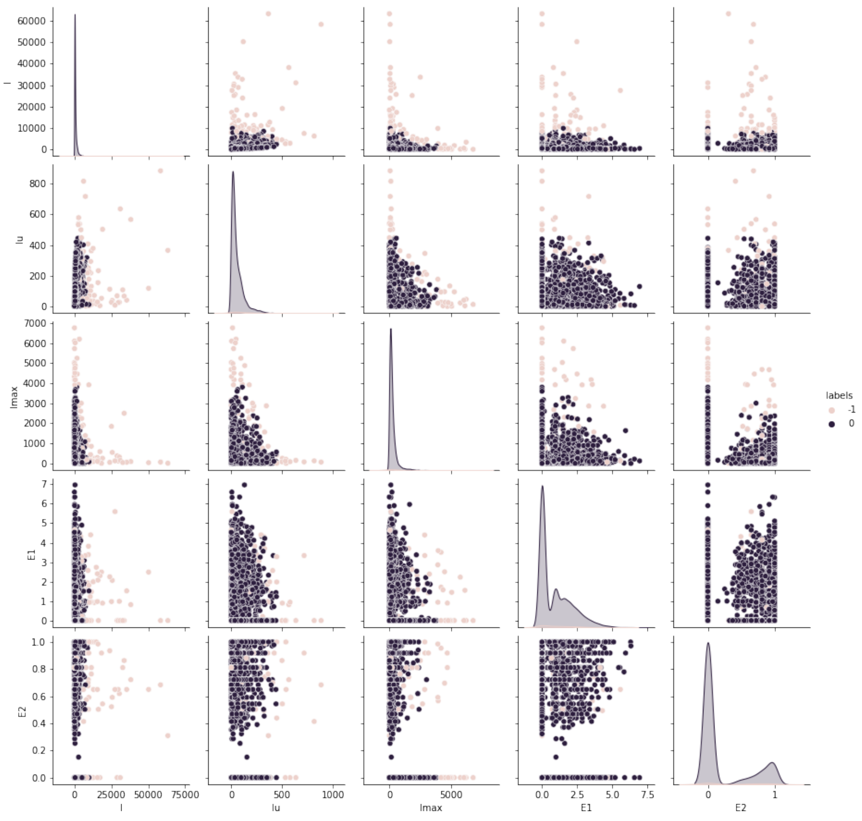
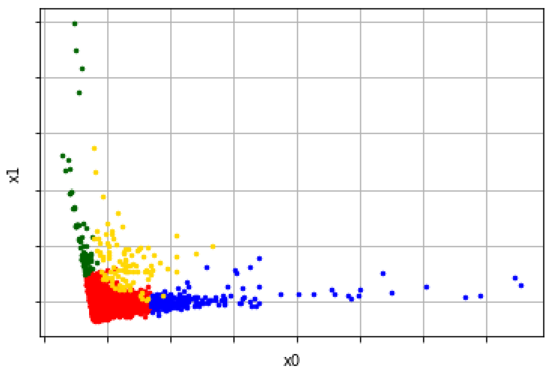


Figure 14. Pairwise plot of mandatory features after DBSCAN clustering

The first issue we noticed is that in our dataset the algorithm found only two clusters. The first targeted with 0 and the second target with -1 that represent outliers. This happens because in the data there are not different high-density regions, but only one large high-density region. For this reason, after DBSCAN clustering we found only one "real" cluster (excluding outliers cluster). In fact, also when we evaluate the goodness of this clustering method, we don't achieve better results in term of Silhouette score and Separation score.

2.4. Alternative Clustering Techniques

All previous algorithm have been implemented via scikit-learn library, and, as a more deep clustering analysis, we decide to apply also some methods from the pyclustering library in order to compare the given results. Pyclustering is a Python, C++ data mining library that provides several algorithms for clustering analysis, oscillatory networks and neural networks. Concerning the clustering analysis there are a huge amount of algorithms, but we try to implements only X-Means and G-Means algotirhms. X-Means clustering algorithm is an extended K-Means which tries to automatically determine the number of clusters based on BIC scores. Starting with only one cluster, the X-Means algorithm goes into action after each run of K-Means, making local decisions about which subset of the current centroids should split themselves in order to better fit the data. The splitting decision is done by computing the Bayesian Information Criterion (BIC). G-Means, instead, uses a statistical test to decide whether to split a K-Means center into two centers in order to determine an appropriate amount of cluster. The Anderson-Darling statistical test for a Gaussian distribution is used to make this splitting decision. However, since the G-Means algorithm provided by the pyclustering library does not allow to specify a maximum amount of clusters, it gives as result a number of clusters greater than two hundreds, for this reason we decide to not reporting it. However, concerning X-Means, we propose a plot visualization of the four main clusters considering that PCA analysis have been performed in order to extract the top three principal components.



(a)

Figure 15. Plot visualization of the four main clusters considering that PCA analysis have been performed in order to extract the top three principal components.

3. Predictive Analysis

3.1. Feature extraction / Data preparation

In the first phase of the predictive analysis task, we prepared the dataset by adding features and removing others, to have a better starting point on which to make predictions by classification. We added three new features related to how many items of each type a customer has purchased:

- **Icheap**: number of cheap items bought by the customer.
- **Iavg**: number of average cost items bought by the customer.
- **Iexp**: number of expensive items bought by the customer.

Then we computed the **CustomerType** label used as the target for the predictions. We defined each type of customers using the inter-quartile distance of the average cost of the cart for each customer. Using this approach, we uniformly distribute each type of customer.
Regarding the removed features, we dropped from the dataset the features used for the calculation of CustomerType, TotalCost and TotalCart, and we also dropped some useless features for the predictive

analysis as the entropy features, E1 and E2, and then CustomerID. A categorical attribute remained in the dataset, the Country value. We discretized it, getting all numerical attributes to perform the classification task.

3.2. Model description and comparison

In the predictive analysis task, we have trained and evaluated six different models. All models were taken from Python's libraries *scikit-learn* and *tensorflow*. We split the data into two parts: the training set (70% of the dataset) with whom we trained each model and the test set (30% of the dataset) with whom we evaluated each model. As metrics for the evaluation, we mainly used test score and train score. In the case that we had very similar results, we also used $Precision = \frac{TruePositive}{TruePositive+FalsePositive}$, $Recall = \frac{TruePositive}{TruePositive+FalseNegative}$ and $F1score = \frac{2\cdot(Precision\cdot Recall)}{Precision+Recall}$. Below there is a quick description of the tested models, how we initialized them, and how we have done the hyperparameters tuning. In the next section, we will talk more precisely about the chosen model.

- **Decision Tree**: the standard implementation of *DecisionTreeClassifier*. We used as *criterion* for the evaluation of splits the Gini index. Then we done a randomized search for these parameters:
  - *max_depth*: the maximum depth of the tree.
  - *min_samples_leaf*: the minimum number of samples required to be at a leaf node.
  - *min_samples_split*: the minimum number of samples required to split an internal node.
- **AdaBoost**: the implementation of *AdaBoostClassifier*. We used as *base_estimator* a Random Forest. Then we done a targeted search for these parameters:
  - *n_estimators*: the maximum number of estimators.
  - *learning_rate*: the learning rate contribution of each classifier.
- **Gaussian Naive Bayes**: the standard implementation of *GaussianNB*. For this model, we did not do hyperparameters tuning.
- **Random Forest**[3]: the implementation of *RandomForestClassifier*. We fixed as *criterion* for the evaluation of splits the Gini index. Then we evaluated other parameter values:
  - *n_estimators*.
  - *max_features*: the number of features to consider when looking for the best split.
  - *max_depth*.
  - *min_samples_leaf*.
  - *min_samples_split*.
- **K-Nearest Neighbors**: the implementation of *KNeighborsClassifier*. We evaluated three parameters for this estimator:
  - *algorithm*: the type of algorithm used for the Nearest Neighbors.
  - *n_neighbors*: the number of neighbors to use fro the classification.
  - *p*: the power parameter for the Minkowski metric.
- **Multi Layer Perceptron**: detailed in the next section.

In Table 7 there are the evaluations of the models that we tested, using the metrics previously described.

Table 7. Evaluation of models used in the predictive analysis task

Model	Test score	Train score	Precision	Recall	F1 score
Multi Layer Perceptron	90%	93%	89%	90%	90%
AdaBoost	87%	98%	89%	85%	87%
Random Forest	87%	93%	87%	85%	86%
Decision Tree	84%	97%	86%	85%	85%
KNeighbors	69%	75%	71%	65%	67%
Gaussian Naive Bayes	46%	47%	53%	50%	43%

3.3. Proposed Model: Multi Layer Perceptron

Amongst the developed models for this multi-classification task we finally decided to choose the Multi Layer Perceptron (MLP). The model consists in a simple neural network having 10 input features and two hidden layers with a softmax classication output layer, using categorical cross-entropy as the loss function for the training phase.

Starting from the labeled dataset (5567 samples), we randomly left out 20% of it (1114 samples) in order to leave model selection phase unaffected by it. We will refer to the latter from now on as Internal Test Set (ITS). The remaining 80% (4453 samples) has been furtherly divided in Training Set ($Train_{inn}$) and Validation Set (VL) undergoing a k-fold cross validation procedure [4] with $k = 4$ as depicted in Figure 16.

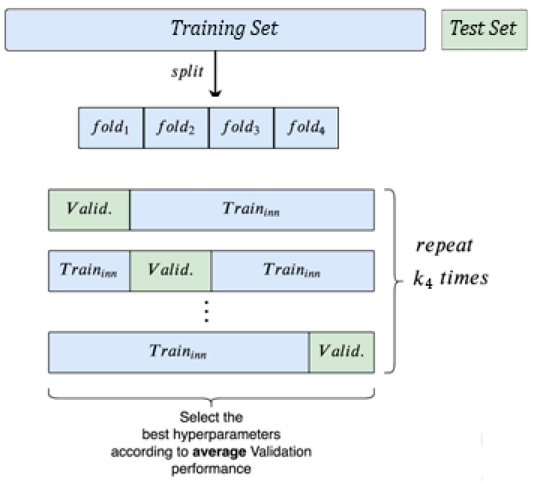

Training Set
Test Set
split
fold1
fold2
fold3
fold4
Valid.
Traininn
repeat k4 times
Select the best hyperparameters according to **average** Validation performance

Figure 16. Own workflow visualization of model selection phase and dataset splitting procedure.

As a preliminary analysis of the hyper-parameters needed for the model selection phase, we investigated the differences amongst different batch sizes (12, 24, 32). The latter was not impacting the performance of the model, but it's been a good trade-off in terms of efficiency in training time. We performed a grid search approach for the model selection phase exploring different learning rates ($1 \times 10^{-1}, 1 \times 10^{-2}, 1 \times 10^{-3}$), model complexity in terms of hidden neurons (50, 100, 150) and number of epochs (100, 200, 300).

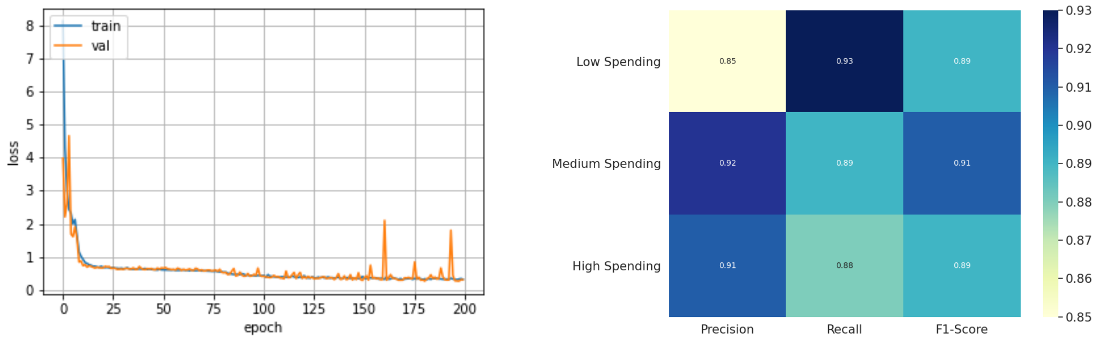


Figure 17. Analysis for the selection of a proper value for ϵ

4. Sequential Pattern Mining

In this section, we are considering the sequential pattern mining problem, which consists of finding statistically relevant patterns among data where the values are provided in a sequence. In order to address this task, we built a data set by modelling the customer as a sequence of baskets. In other words, for each customer there is a sequence of baskets and for each basket there are the bought products. Our goal can be formalized as follow: discover the hidden relationships between products and baskets in order to extract discriminatory behaviors.

In the following section, we will describe the approach we decide to adopt, which is based on the product description generalization, in order to solve the problem of sequential pattern mining.

4.1. Product descriptions generalization

We deeply analyzed the set of product descriptions composed by 3980 unique records, realizing that lowering this uniqueness could lead to the potential retrieval of a larger and more interesting list of patterns for sequence mining tasks.

The proposed approach for generalizing each product description without losing their meaning, consists in normalizing each label according to natural language properties, then computing a distance matrix for each label and finally performing agglomerative clustering in order to reduce this set. We provide an example of the process applied to the actual dataset used for subsequent and previous tasks.

4.1.1. Proposed approach

For instance, if we analyze the small set P containing three product description, the first P_1 contains details about the design which can be left out, also in the second one and the third specifying the color of the product is not providing any additional information.

```
P = [
     'LUNCH BAG I LOVE LONDON' ,
     'LUNCH BAG RED RETROSPOT',
     'LUNCH BAG WITH CUTLERY RETROSPOT'
    ]
```

Initially, we perform a normalization step producing vector P_N, for each product description in the sense that we remove color informations, numbers usually referring to numbered sets of products and english stopwords. As a last step for the normalization process we perform PoS tagging analyzing each single word and removing adjectives, congiunctives, numeral and everything which is not a meaningful noun.

```
PN = [
     'LUNCH BAG LONDON' ,
     'LUNCH BAG RETROSPOT',
     'LUNCH BAG CUTLERY RETROSPOT'
     ]
```

Starting from these new product descriptions, we can build a distance matrix amongst each of them for the agglomerative clustering procedure. Exploiting the symmetry for the used metric being the Jaro-Winkler, the computational cost of building it will be:

$$C = n + n - 1 + n - 2 + ... 1 = n^2 - 1 - 2 ... \; - n = \frac{2n^2 - n^2 - n}{2} = \frac{n^2 - n}{2} \tag{5}$$

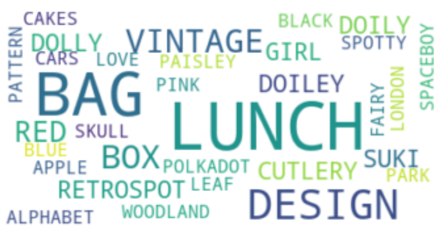
where the cost of computing a single distance is dictated by its equation:

$$sim_j = \left\{ \begin{array}{ll} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + t\right), & \text{otherwise} \end{array} \right\} \qquad sim_w = sim_j + \frac{l(1 - sim_j)}{10} \tag{6}$$

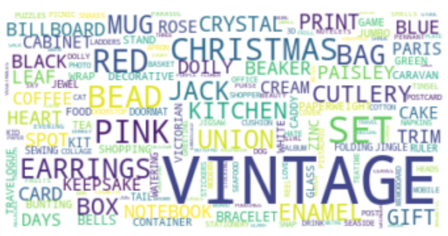
where $|s_j|$ is the lenght of the string, m is the number of matching characters, t half the number of transpositions.

After building the distance matrix and running the agglomerative clustering procedure, we end up with 253 centroids representing unique product description. Since we started with 3980 strings we have a reduction of approximately 93,6%, maintaining a good quality of the residual information.

In Figure 18 we provide two Wordcloud representation of each product description in a cluster, specifically in Figure 18.a the cluster is 'LUNCH BAG' and each word surrounding the centroid contains redundant informations concerning what is inside the lunch bag or its design. A similar but more complex example is depicted in Figure 18.b where each small word compared to the cluster centroid being 'VINTAGE' represents the essence of the transaction, but can be included in a broader set being the one of vintage purchases.



(a)



(b)

Figure 18. (a) Wordcloud plot for the centroid 'LUNCH BAG', **(b)** Wordcloud plot for the centroid 'VINTAGE'

4.2. Experimental results

The overall task of the experiments we ran can be viewed and analyzed from two different prospective:

1. **Frequent item mining**: allows us to analyze the preferred products by customers during each single shopping session;
2. **Association rules mining**: allows us to analyze the relationships between products among different shopping sessions in the medium to long term. This task is interesting because it allows to better understand the purchasing needs based on what has already been bought and also if there are recurring patterns linked to specific product categories.

For both, we tried different algorithms from different packages in Python and we compared their results in terms of patterns found and execution time. Now let's take a closer look at how these algorithms work and how we used them for our purposes.

4.2.1. Frequent Itemset Mining

The Frequent Itemset Mining discovers sub-sequences that are common to more than minimum support (*minsup*) sequences in a sequence. A sequence is just an ordered list of transactions. The *minsup* can be defined in terms of the number of transactions or in percentage related to all transactions in the sequence. For some of the algorithms that we tested it is possible to add further constraints such as the minimum or maximum length of the patterns, or the maximum depth in searching for patterns. We compared three packages: *prefixspan*, *gsppy* and *spmf*. All of these retrieve the same results, but they are very different in terms of time. The first two are slower than *spmf*, but they are easier to use. *Spmf* is the most complete and fastest than others, but it requires input in a specific format. Moreover, it implements a lot of mining algorithms that can be used once you understand how to define the input.

- **PrefixSpan**: it is a very simple library to use. It contains just a method to initialize the constructor with the sequence of transactions of which we want to find the frequent itemsets. Then we define the *minsup* in terms of transactions and we can retrieve the frequent itemsets or the top-k itemsets in our sequence. *PrefixSpan* also allows inserting filters for the mining such as the length of patterns or more complicated filters.
- **GSPPy**: it is the simplest, but the slowest that we tested. Just execute the mining by giving the sequence and the *minsup*, in percentage, as input.
- **SPMF**[5]: it is the fastest and the most complete library. The library is written in Java, but we used a wrapper for Python. To use *SPMF* we encode all products with a unique number and we had to order increasing all products in carts. It also implements a lot of algorithms that we tested for the sequential pattern mining and they differ for the execution time and the memory used. The *minsup* for all algorithms is in percentage. The algorithms that we tested are:
  - *PrefixSpan*.
  - *GSP*.
  - *SPADE*.
  - *SPAM*: is the only one that allows to define the pattern size in addition to the *minsup*.
  - *LAPIN*.

In Table 8 the measurements of execution time and memory usage for all *SPMF* algorithms. The measurement are referred to a dataset with 2462 customer with 2 carts per customer on average, and each cart with 15 products on average. The measurements taken may be slightly different on different machines.

Table 8. Evaluation of SPMF algorithms in execution time and memory usage

Algorithm	Time	Memory
PrefixSpan	750 ms	50 Mb
GSP	5600 ms	100 Mb
SPADE	290 ms	35 Mb
SPAM	890 ms	40 Mb
LAPIN	1300 ms	40 Mb

In Figure 19 we can see some results obtained from the frequent itemset mining. The figure contains the top-20 frequent itemsets with $min\_support \geq 10\%$ discovered with SPADE algorithm of *SPMF*. From this chart we can see that the most frequent patterns are of size one, but there are some frequent patterns with size two, such as {heart, retrospot} or {heart, vintage}. The macro-category of products *heart*, *retrospot* and *vintage* are the most frequent in our dataset. This means that the vast majority of purchased products belong in these categories.

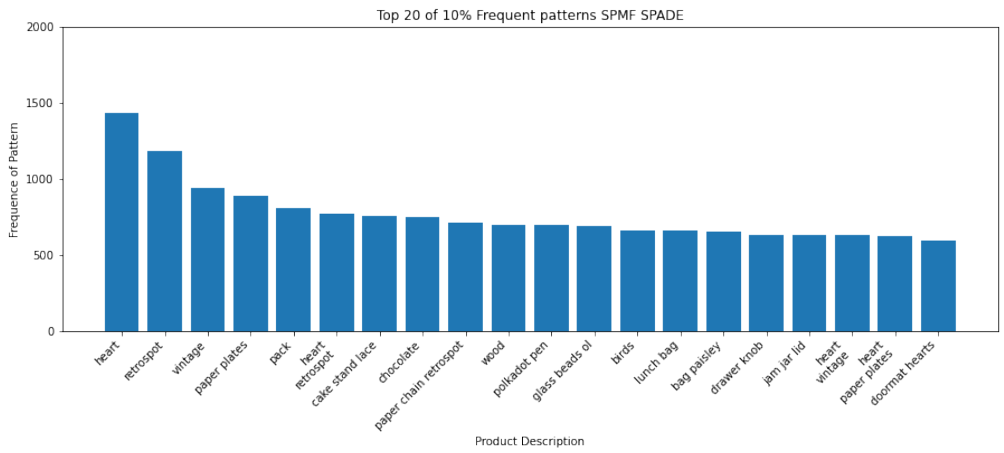


Figure 19. Most frequent itemset from SPADE algorithm with $min\_support \geq 10\%$

4.2.2. *Association Rules Mining*

A Frequent itemset mining approach can be useful to detect common purchasing behaviours related to a single shopping cart, but we are also interested in spotting patterns amongst different shopping sessions.

An association rule in our dataset is defined as an implication of the form:

$$X \Rightarrow Y\text{, where } X, Y \subseteq products \text{ and } t(Cart(X)) < t(Cart(Y)) \tag{7}$$

We analyze the importance of each rule by two parameters: support and confidence.

Support is an indication of how frequently the association appears in the dataset. The support of an association rule $X \Rightarrow Y$ with respect to a set of transactions T is defined as the proportion of transactions t in the dataset which contains the association rule. Instead, we can express the confidence as an indication of how often the rule has been found to be true.

$$support(X \Rightarrow Y) = \frac{|t \in T; X \Rightarrow Y \subseteq t|}{|T|} \qquad confidence(X \Rightarrow Y) = \frac{support(X \cup Y)}{support(X)} \tag{8}$$

To find association rules in our dataset we used a Python package called *apyori*, that is very easy to use. It takes in input the list of transactions and allows us to set the minimum support and the minimum confidence, in percentage, for the rules that we are mining. It just returns a list of all rules with relative support and confidence. In Table 9 there are some of the association rules found by *apyori* with $min\_support \geq 10\%$ and $confidence \geq 50\%$.

Table 9. Association rules found by Apyori with relative support and confidence

Rule	**Support**	**Confidence**
$retrospot \Rightarrow heart$	21%	58%
$vintage \Rightarrow heart$	17%	64%
$paperplates \Rightarrow heart$	17%	64%
$paperplates \Rightarrow retrospot$	15%	58%
$chocolate \Rightarrow heart$	15%	72%
$vintage \Rightarrow retrospot$	15%	56%
$pack \Rightarrow heart$	14%	64%
$glassbeadsol \Rightarrow heart$	14%	70%

4.2.3. Time constrained association rules

We implemented a new version from the provided implementation of Generalized Sequential Pattern (GSP), called TGSP (with the addition of time constraints), based on the apriori principle to meet our objectives. Our implementation only requires a list of timestamps referring to the purchase date as long as the list of events occurring for each customer and provides three main time constraints to tune for the analysis:

- **Minimum gap**: we usually set this parameter as 1 day, since we are not interested into breaks between shopping sessions.
- **Maximum gap**: we are interested into pruning association rules being too far from each other in terms of purchase date. A fair trade-off is represented by 3 to 7 days.
- **Minimum interval**: we are also interested in the lenght of a timeframe in which the association rules should fall. A minimum period of observation of 3 to 6 months usually provides interesting results on the customer shopping behaviour.

Starting from the original customers, we retrieved only those over a *minimum interval* of 3 months of purchases, by doing so, we prune all the occasional buyers. We set as one week the *maximum gap* parameter defining the maximal distance allowed amongst instances of an association rule.

For each customer we calculated a value resulting in a label belonging to three different classes: 'high-spending', 'medium-spending' and 'low-spending', for all of them we ran the TGSP algorithm with the above-mentioned time constraints.

In Table 10 we report the association rules found for low-spending customers with a minimum support of 20%. The cluster *heart* refers to all heart-shaped products in the dataset, while *vintage* and *retro* contain products originating from a previous era. As we can see from the results there is a common habit exhibited by customers in buying vintage or retro items consequently to heart-shaped ones. Another strong habit is the one of persistently buying vintage items, this is explainable in the sense that people buying specific categories of products usually buy them back.

Table 10. Association rules found by running TGSP algorithm for low-spending customers from the dataset (min_gap = 1 day, max_gap = 7 days, min_interval = 3 months)

Association Rule	**GSP Support**	**TGSP Support**
$heart \Rightarrow vintage$	22%	3%
$heart \Rightarrow retro$	21%	3%
$vintage \Rightarrow heart$	21%	3%
$vintage \Rightarrow vintage$	20%	2%

The patterns emerged from the *low-spending* customers analysis reoccurred also for *medium-spending* and *high-spending* customers, having a stronger support percentage.

Table 11. Association rules found by running TGSP algorithm for medium-spending customers from the dataset (min_gap = 1 day, max_gap = 7 days, min_interval = 3 months)

Association Rule	GSP Support	TGSP Support
heart ⇒ *vintage*	44%	3 %
retro ⇒ *retro*	44%	3 %
vintage ⇒ *vintage*	43%	4%
vintage ⇒ *heart*	42%	4%
retro ⇒ *heart*	40%	3 %
heart ⇒ *retro*	39%	3%
retro ⇒ *vintage*	38%	3%
vintage ⇒ *retro*	35%	3%

Table 12. Association rules found by running TGSP algorithm for high-spending customers from the dataset (min_gap = 1 day, max_gap = 7 days, min_interval = 3 months)

Association Rule	GSP Support	TGSP Support
vintage ⇒ *vintage*	58%	5%
retro ⇒ *retro*	55%	5 %
retro ⇒ *vintage*	55%	5%
heart ⇒ *vintage*	53%	4 %
retro ⇒ *heart*	52%	5 %
vintage ⇒ *heart*	51%	3%
vintage ⇒ *retro*	49%	3%
heart ⇒ *retro*	47%	4%
chocolate ⇒ *heart*	42%	2%

References

1. Satopaa, V.; Albrecht, J.; Irwin, D.; Raghavan, B. Finding a "Kneedle" in a Haystack: Detecting Knee Points in System Behavior. 2011 31st International Conference on Distributed Computing Systems Workshops, 2011, pp. 166–171. doi:10.1109/ICDCSW.2011.20.
2. Sharma, A.; Sharma, A. KNN-DBSCAN: Using k-nearest neighbor information for parameter-free density based clustering. 2017, pp. 787–792. doi:10.1109/ICICICT1.2017.8342664.
3. Koehrsen, W. Hyperparameter Tuning the Random Forest in Python, 2018.
4. Kohavi, R.; others. A study of cross-validation and bootstrap for accuracy estimation and model selection. Ijcai. Montreal, Canada, 1995, Vol. 14, pp. 1137–1145.
5. Fournier Viger, P.; Lin, C.W.; Gomariz, A.; Gueniche, T.; Soltani, A.; Deng, Z.H.; Lam, H. The SPMF Open-Source Data Mining Library Version 2. 2016, Vol. 9853, pp. 36–40. doi:10.1007/978-3-319-46131-1_8.