Data Mining Project

Giulio Purgatorio - 516292
Jacopo Massa - 543870
GROUP 5

Academic Year 2020/21

Contents

1 Data Understanding **1**
1.1 Data semantics . . . 1
1.2 Data distributions and statistics . . . 1
1.2.1 Time-related . . . 1
1.2.2 Geolocalization-related . . . 2
1.2.3 Products-related . . . 2
1.2.4 Extra attributes . . . 2
1.3 New features . . . 3
1.4 Outliers . . . 3
1.5 Correlation . . . 4

2 Clustering **5**
2.1 K-Means . . . 5
2.1.1 Parameters and distance function . . . 5
2.1.2 Cluster analysis . . . 5
2.2 DBSCAN . . . 7
2.2.1 Parameters and distance function . . . 7
2.2.2 Cluster analysis . . . 7
2.3 Hierarchical clustering . . . 8
2.3.1 Parameters and distance function . . . 8
2.3.2 Dendrograms analysis . . . 8
2.4 Evaluation and comparison of clustering approaches . . . 9

3 Classification **10**
3.1 Model Selection . . . 10
3.2 Decision Tree . . . 11
3.3 Random Forest . . . 12
3.4 SVM . . . 12
3.5 KNN . . . 13
3.6 Naive Bayesian . . . 14
3.7 Comparison of used predictive models . . . 14
3.7.1 Fit Time . . . 14
3.7.2 Accuracy . . . 15
3.7.3 Conclusions . . . 15

4 Sequential Pattern Mining **16**
4.1 Sequence Generation . . . 16
4.2 GSP Results . . . 16

5 Association Rules Mining **17**
5.1 Frequent Patterns . . . 17
5.2 Association Rules . . . 18

Chapter 1

Data Understanding

The proposed dataset contains **471910** transactions carried out by some customers in a supermarket. Each transaction is described by a set of **8 attributes** (originally), which provide information on both the customer and the involved products. All the statistics and considerations were made before the data cleaning phase.

1.1 Data semantics

This section contains a brief description of each attribute that appears in the original dataset.

Attribute	**Type**	**Description**
BasketID	`Numerical`	Unique alphanumeric ID assigned to a *basket*, defined as a set of transactions.
BasketDate	`Numerical`	The date on which the transaction was made.
Sale	`Numerical`	Cost of one unit of product.
CustomerID	`Numerical`	The ID of the customer.
CustomerCountry	`Categorical`	The Customer's country of origin.
ProdID	`Numerical`	The ID of the product.
ProdDescr	`Categorical`	A description of the product.
Qta	`Numerical`	The number of units bought in the transaction.

1.2 Data distributions and statistics

Since we need to define a customer's profile, we modified the original dataset removing all the records containing null values in the `CustomerID` attribute: these can't contribute to our goal and it also helps reduce the size of the dataset (**65080** transactions without `CustomerID`). Further testing revealed that also **753** transactions had a missing `ProdDescr` attribute, but these were all included in the previous ones. The cleaning led us to a total of **406830** records.
Then we analyzed the dataset and saw the presence of many records that wouldn't help achieve the main target of this study (i.e. the presence of null `Sales` and negative `Qtas`). The first doesn't benefit our studies because "zero-sales" are not economic-relevant to define a customer's profile. Instead, the amount `Qta` of sold items, when negative, might mean many things: we assumed it's a returned item *(due to many possible reasons, like broken, defective, etc.)* because there were a lot of bought and then later "returned" items, which isn't related in any way to how a customer behaves. This led to the removal of **1279** records in the first case, and **9752** in the second one.

1.2.1 Time-related

The stored records range from the ***Minimum date:*** *2010-12-01 08:26:00* to the ***Maximum date:*** *2011-12-09 12:50:00*, so just a bit longer than a whole year. As shown in Figure 1.1, the sales are kind of constant throughout the year and increased in the proximity of Christmas. In fact, the ***day with more sales*** *was 2011-12-05* and the ***day with fewer sales*** *was 2010-12-22*. A simple way to justify this would be that the store started its activity that first Christmas and steadily increased its clientele from time to time.

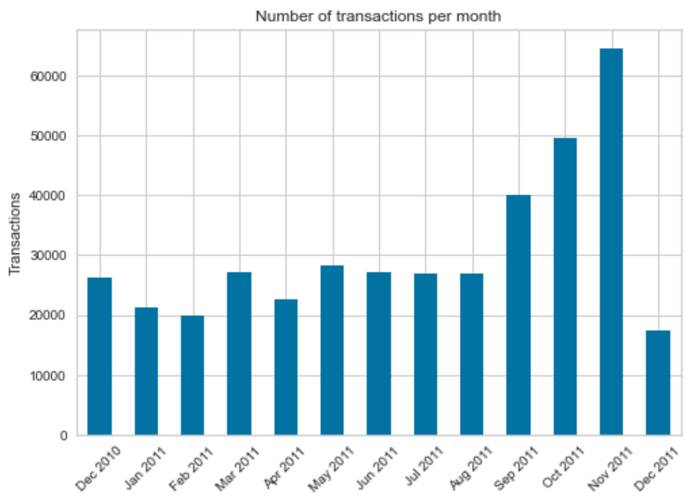


Figure 1.1: `BasketDate` distribution.

1.2.2 Geolocalization-related

As shown in Figure 1.2, most of the customers come from the United Kingdom. This suggests that the store is probably located there, but no customer-side information is relevant here and this attribute will be no longer considered for this reason.

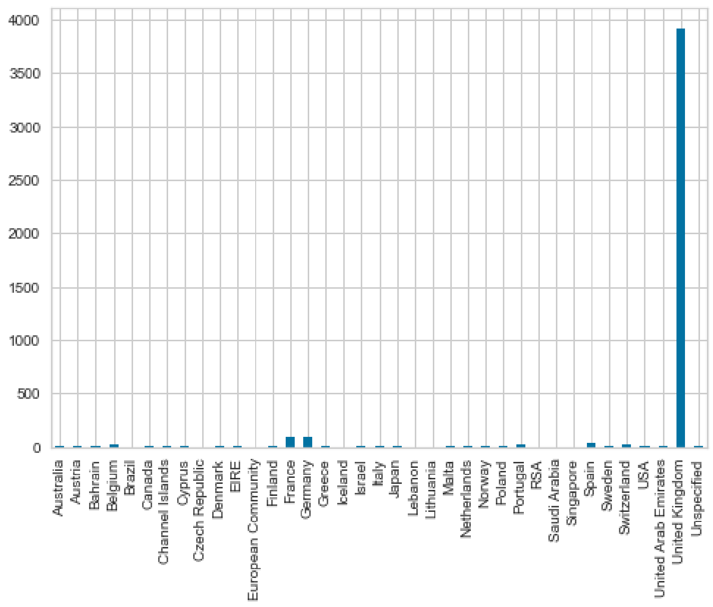


Figure 1.2: `CustomerCountry` distribution.

1.2.3 Products-related

There are a total of **3665** different products that were sold. Simple analyses are:

- **Best seller:** the best-selling item. ID: 85123A, WHITE HANGING HEART T-LIGHT HOLDER.
- **Worst seller:** the worst-selling item. ID: 22323 PINK POLKADOT KIDS BAG.

1.2.4 Extra attributes

We also added a new numerical attribute to help us make some studies:

Amount	`Numerical`	The amount of the transaction, calculated by multiplying the `Sale` by the `Qta`.

1.3 New features

To describe in a better way the customer profile and also improve data quality, we extracted some features for each customer, creating a new dataset indexed by their `CustomersIDs`.

Attribute	**Type**	**Notes**	**Mean**	**Std**	**Min**	**Max**
I	`Numerical`	The *total* number of items purchased by a customer during the period of observation.	91.72	228.78	1.00	7847.00
Iu	`Numerical`	The number of *distinct* items bought by a customer in the period of observation.	61.50	85.36	1.00	1787.00
Imax	`Numerical`	The *maximum* number of items purchased by a customer during a shopping session.	32.08	31.24	1.00	542.00
Entropy	`Numerical`	The *Shannon entropy* on the purchasing behavior of the customer.	3.43	1.28	0.00	8.33
BasketNum	`Numerical`	The number of distinct baskets for each customer.	4.27	7.69	1.00	209.00
SumExp	`Numerical`	The total expenditure for each customer.	2054.27	8989.23	3.75	280206.00
AvgExp	`Numerical`	The entire expenditure for each customer divided by the number of baskets.	355.00	329.02	3.45	4932.13

1.4 Outliers

The last step for data quality assessment was the outliers detection, which was made only for the new dataset (the one with all the new attributes).
For outliers' analysis and removal, we decided to use the *Z-Score* metric, which is an important measurement that tells how many Standard Deviation above or below a number is from the mean of the dataset.
Following the *empirical rule*, we considered as outliers all the data that had an absolute value of Z-score above 3, removing them.

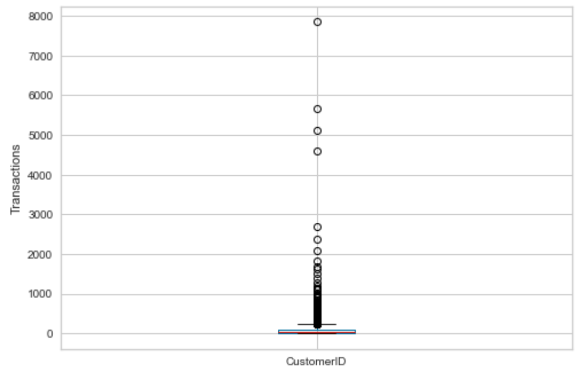


Figure 1.3: Boxplot of customers' number of transactions.

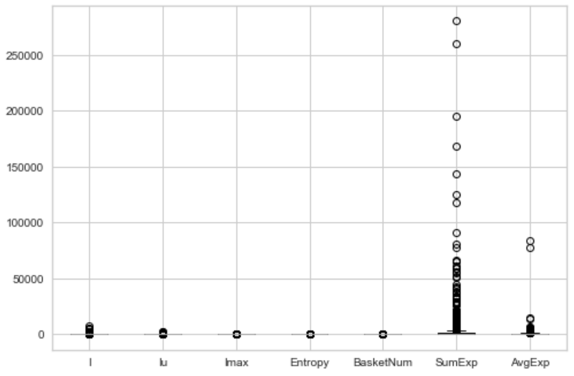


Figure 1.4: Boxplot for the new dataset.

Once outliers were removed, the total number of customers has dropped down from **4338** to **4171** (**167** "outliers" customers).

1.5 Correlation

As shown in Figure 1.5, correlation in the original dataset isn't high in most of the considered pairs. Exceptions are:

- `BasketIDs` and `BasketDates`: all the transactions that belong to the same basket are made on the same date.
- `ProdID` and `ProdDescr`: the same item (usually) has the same description.

So, due to the high correlation score (**0.98**) for descriptions and items, we can safely assume that `ProdDescr` is a superfluous attribute and so it can be dropped in future studies.
On the other hand, our new attribute `Amount` has of course a very high correlation with the `Qta` attribute, simply because they're directly proportional.

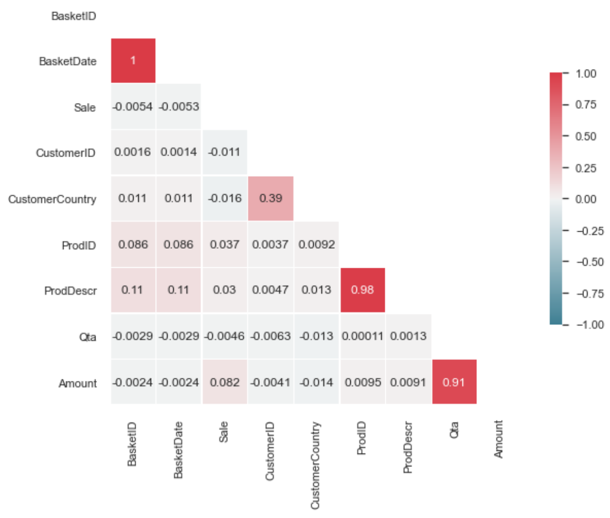


Figure 1.5: Original dataset correlation.

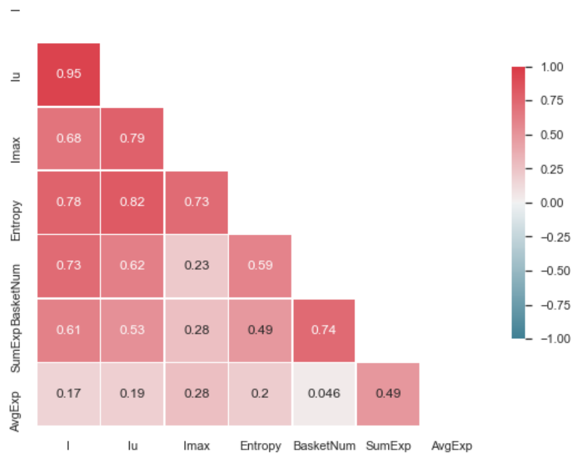


Figure 1.6: New dataset correlation.

In our new dataset, all the numeric attributes are strictly correlated as shown in Figure 1.6. Two main examples are:

- `Iu` and `I`, because they both refer to bought items.
- `SumExp` and `BasketNum`, because the more a customer comes to the store, the more he will eventually spend.

For reference, a standard *describe()* has been called on the new dataframe and results can be seen in the Table below.

Attribute	**Mean**	**Std**	**Min**	**Max**
I	70.31	86.62	1.00	756.00
Iu	52.37	53.55	1.00	315.00
Imax	28.87	22.91	1.00	125.00
Entropy	3.35	1.22	0.00	6.33
BasketNum	3.58	3.80	1.00	27.00
SumExp	1327.56	2058.23	3.75	28754.11
AvgExp	355.00	329.02	3.45	4932.13

Table 1.1: Description of new dataframe.

Chapter 2

Clustering

It's a good practice in clustering to normalize data to avoid biases. Two possible approaches are the `StandardScaler` (also called `Z-Score`) and the `Min-MaxScaler`. We used the last one because the obtained results were more interesting.
We were able to use our whole new dataset because every attribute is numerical and has a very high pairwise correlation, as explained previously in Section 1.5.

2.1 K-Means

2.1.1 Parameters and distance function

The main parameter for `K-Means` is the number of clusters. To choose this value we used the *Elbow* method, as shown in Figure 2.1 that shows us that the best value is **5**.

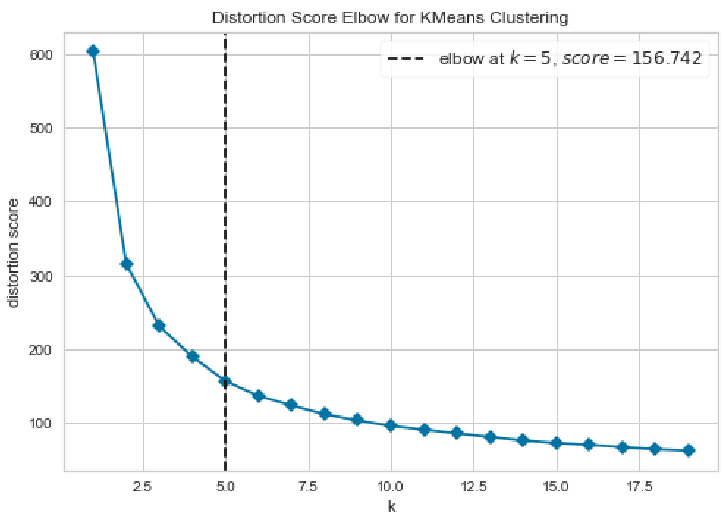


Figure 2.1: Elbow method to find the best value of k.

2.1.2 Cluster analysis

It's possible to see in Figure 2.2 the 2D visual representation of the 5 returned clusters with their respective centroids.

We've also used **Parallel coordinates** in order to find patterns: the plot is shown in Figure 2.3. In our dataset, which has 7 numerical attributes and 5 different classes representing the clusters, we can see that all the lines are either parallel or at least concordant in all adjacent pairs, indicating a shared behavior across clusters: for this reason, we can safely say that there's a general pattern among all of them.
Then, we executed the `K-Means` algorithm to find the given number of clusters. Finally, the results are visible in Figure 2.4.

The clusters are divided as follows:

- **Cluster 0** and **Cluster 1** refer to clients that come occasionally to the store and buy few things: specifically *Cluster 0* is about people that buy few things, while *Cluster 1* is about clients that buy more items. The analysis of K-means divided them, but they're very similar in real-life scenarios: they both represent the **occasional customer**.

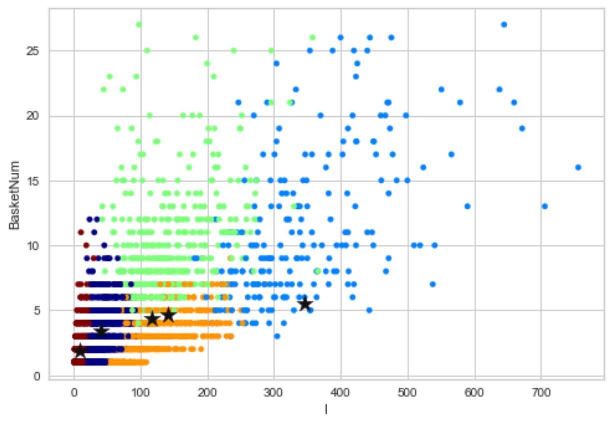


Figure 2.2: Cluster and centroids visualization in two dimensions.

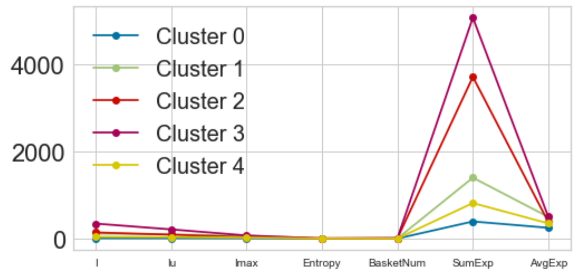


Figure 2.3: Visualization of clusters centers through parallel coordinates.

- **Cluster 2** is similar to the previous ones, but the difference lies in the number of bought items, which is higher. In any case, they opt to buy low-cost items, because the `SumExp` is nearly at the same level as Clusters 0 and 1.
- **Cluster 3** buy nearly the same number of items as *Cluster 2*, but the number of times they come back to the store is higher and they tend to spend more. This cluster describes the one containing **regular customers**.
- Finally, **Cluster 4** describes customers that buy a lot of expensive things and they tend to return to the store frequently. These are the **loyal customers** or simply the “best” ones, from an economic point of view.

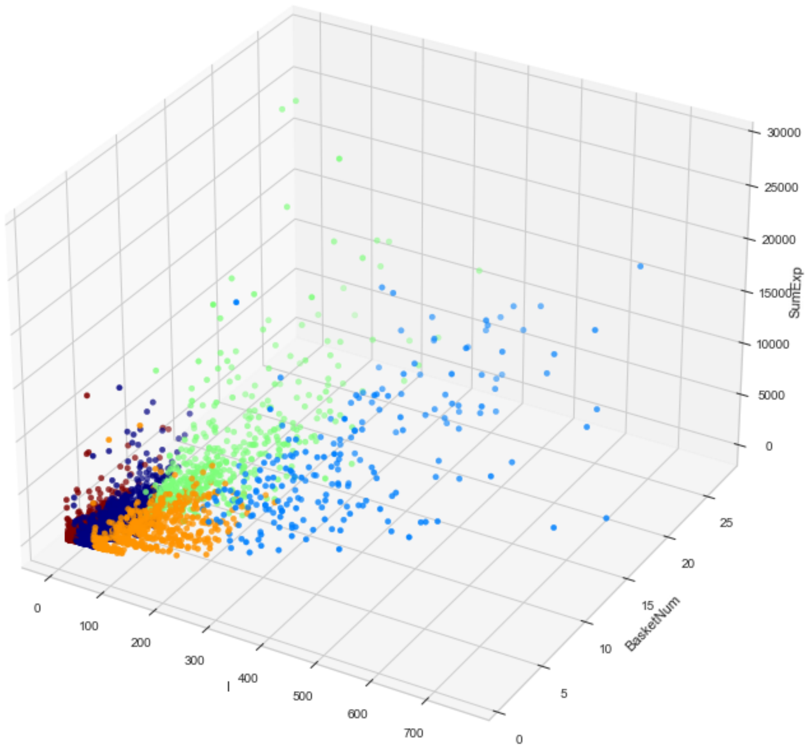


Figure 2.4: Visualization of clusters in 3 dimensions.

2.2 DBSCAN

2.2.1 Parameters and distance function

We still used the `MinMax` scaler and the *Euclidean Distance*, because it's a good metric with numeric values. `DBSCAN` uses these two parameters:

- **eps**: the maximum distance between two samples for them to be considered as in the same neighborhood.
- **min_samples**: the number of samples in a neighborhood for a point to be considered as a core point. This includes the point itself.

Since there's no automatic way to decide the value of these parameters, we did some research and found that a standard setup is to choose *min_samples* as twice the dimensionality of the dataset (`min_samples` = **2*7 = 14**). About *eps*, we used a technique that exploits again the *Elbow* method. This technique calculates the average distance between each point and its *k-nearest neighbors*, where `k = min_samples` value. The average k-distances are then plotted (as in Figure 2.5) in ascending order. We found the optimal value for `eps` at the point of maximum curvature (same as K-Means, but on the y-axis), so at `eps` = **0.351**.

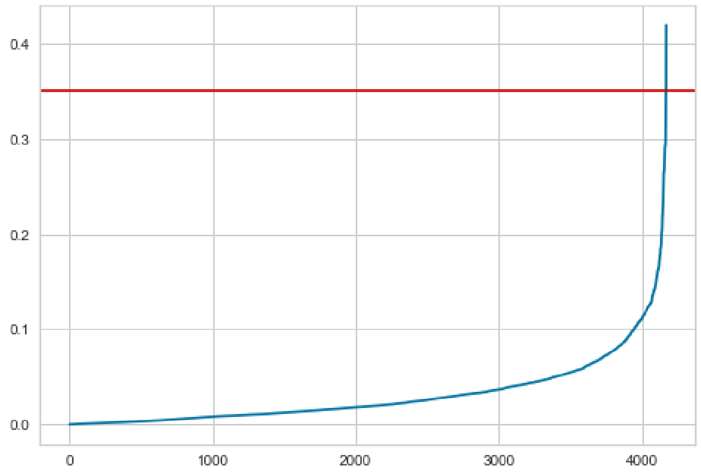


Figure 2.5: Best `eps` value by applying Elbow Method.

2.2.2 Cluster analysis

After having executed the algorithm with the parameters chosen as described before (Section 2.2.1), the algorithm has returned two clusters where one was almost empty (**15** elements) and the other one with all the rest (**4156** elements). After this failure, we also tried to change these parameters by hand, but despite this, neither the number of clusters nor the data distribution changes significantly. Results are visible in Figure 2.6

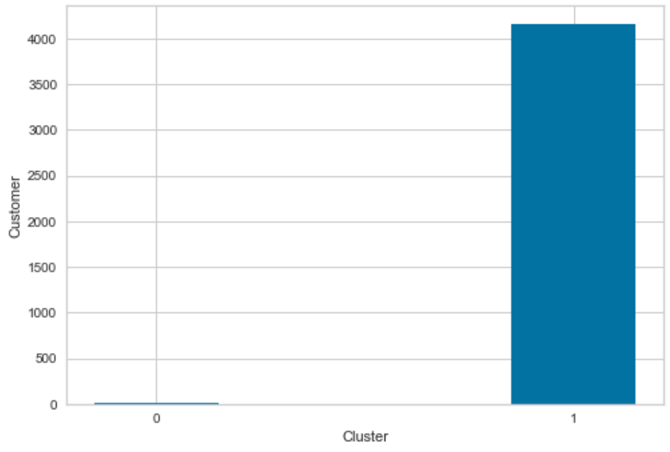


Figure 2.6: Data distribution in DBSCAN Clusters.

2.3 Hierarchical clustering

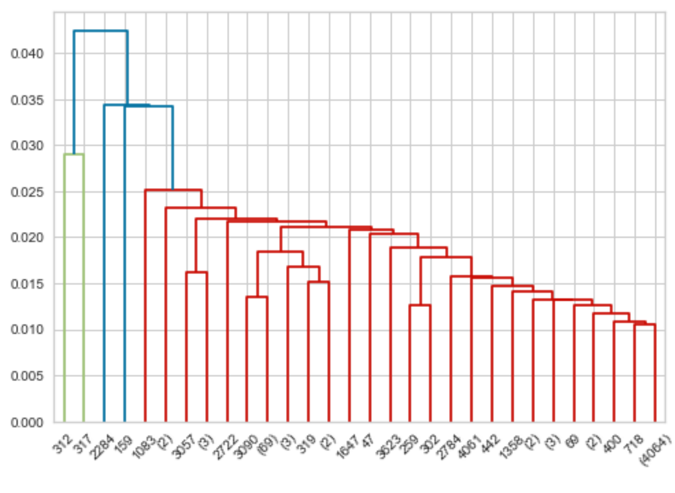
2.3.1 Parameters and distance function

Out of all the possibilities, we've chosen **Single**, **Complete**, **Average**, and **Ward** methods and we applied them to different metrics (**Euclidean Distance, Cosine Similarity, and Manhattan Distance**).

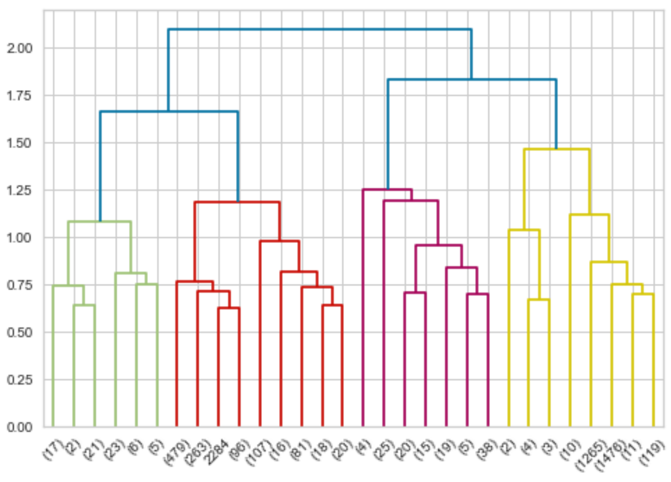
2.3.2 Dendrograms analysis

We've run the hierarchical clustering algorithm with all the different combinations of methods and metrics and we'll now list the most relevant ones for conciseness.

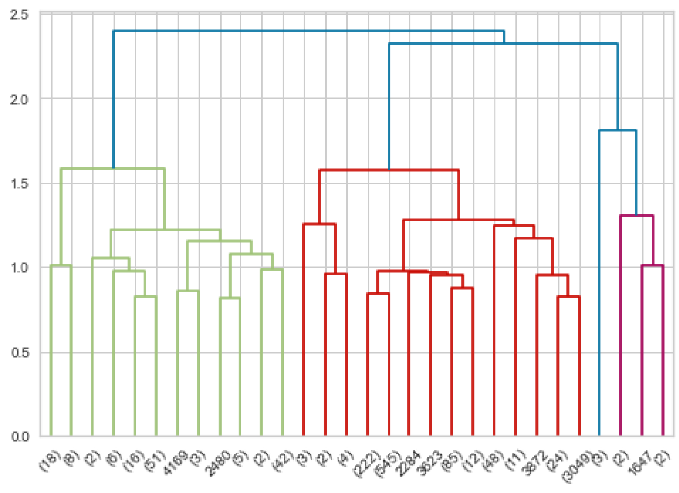
- The **Single method** wasn't a good one because it creates a very large cluster (more than **99%** of items) and spreading the rest in one-element clusters, like what happened in Figure 2.7a where there is a huge cluster of **4167** elements and three tiny clusters with 1 or 2 points.
- The **Complete method** returned very interesting results with the *Euclidean Distance* metric: this confirms that it's a really good metric with numerical values. The resulting dendrogram is shown in Figure 2.7b.
- The **Average method** behaved almost exactly like the Single method, just retrieving a lower number of clusters, as shown in Figure 2.7c.
- Finally, the **Ward method** didn't retrieve a suitable data distribution. As shown in Figure 2.7d, it always found two clusters and in some cases *(like the shown one)* these are almost empty as well.



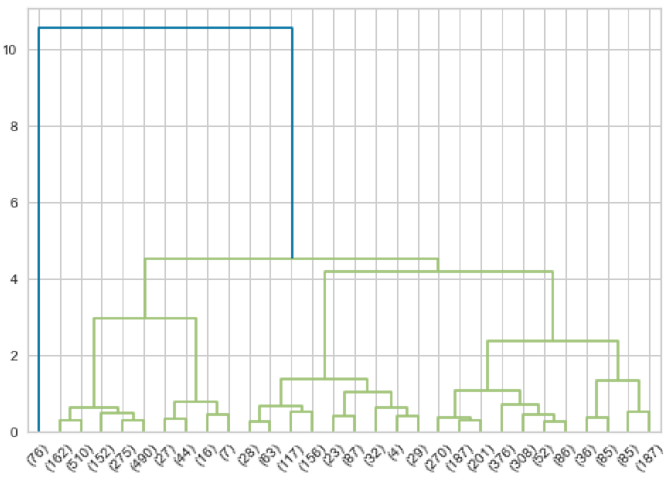
(a) Single - Cosine



(b) Complete - Euclidean



(c) Average - Manhattan



(d) Ward - Cosine

2.4 Evaluation and comparison of clustering approaches

To conclude the clustering section, we'll now compare the different clustering algorithms and the obtained clusters.

K-Means produced a good distribution for our data. As we said in Section 2.1.2, we've obtained **5** clusters that can identify 3 different real-life customer profiles.

With DBSCAN , no matter what the values of the initial parameters, we couldn't obtain a good clustering because it always produced a huge cluster with the majority of the points and another negligible one.

The Hierarchical approach produced very unbalanced clusterings with some linkage methods *(e.g. Single, Ward)*. Instead, when using other combinations (like *Complete Method* with *Euclidean Distance*), we've obtained something similar to the results of the `K-Means` approach that can reflect the previously discussed customer profiles.

So, in summary, `K-Means` and `Hierarchical` approaches returned good clusterings, but we consider more interesting the `K-Means` one.

Chapter 3

Classification

The goal of this part was to characterize each customer with a label that would identify its behavior as a *low/medium/high-spending* one. We thought that the best way to define this was to look at how much a specific customer spent in its entirety, instead of looking for specific transactions that were expensive or not. So, the main idea is that a *high-spending customer* is someone that has spent more than the average: having a low number of *baskets*, even with a very high total cost, may not be enough to be entitled to a specific label.
To achieve this, we used our precedently defined parameter `SumExp`, which is simply the total amount of how much a customer spent.
Labels are nominal as requested, precisely identified by `low`, `medium`, and `high`.
The simplest way to assign the labels was to sort the dataset by the `SumExp` parameter and then adding a `Label` attribute that was set as *low* for the first third of the data, *medium* for the second, and finally *high* for the rest. The fact that there are 3 possible labels means that we're in a *multiclass classification problem.*
Then, to predict classes, we decided to consider the following attributes for the following reasons:

- `I:` the number of items is related to how much someone eventually spends.
- `BasketNum:` the number of times a customer comes to the store is also obviously related to its final expense.
- `Entropy:` the higher this parameter gets, the higher both the number of transactions and the cost of items are going to be.

All other attributes were deleted as they weren't relevant to our case study.
Our pre-processing part then ended with the splitting of the entire dataset into a *train-set* (**70%**) and a *test-set* (**30%**), which are obviously disjoint as the **Holdout method** requires.

3.1 Model Selection

The classification shown in the reports for the techniques in Table 3.1 was done by using as parameters the best values obtained through model selection.
We tried different and randomly chosen parameters' combinations and for each of them, we calculated `Accuracy` and `F1 Score`. We then picked the first one because the class distribution for our data is balanced.
For each model that is shown in Table 3.1, we'll show the top **5** combinations of parameters ordered by `Accuracy`: the highlighted green row has been used to perform the classification, so the shown results refer to it.

Table 3.1: Given parameters for model selection

Algorithm	**Parameters**
Decision Tree	**max_depth**: *[2,3,4,5,6,7,8,9,10,11,12,None]*, **min_samples_split**: *randint(3, 30)*, **min_samples_leaf**: *randint(1, 40)*, **criterion**: *[entropy, gini]*, **splitter**: *[best, random]*
Random Forest	**n_estimators**: *[30,50,100,300,500,750,1000]*, **max_depth**: *[2,3,4,5,6,7,8,9,10,11,12,None]*, **max_features**: *randint(1, len(train_set))*, **min_samples_split**: *randint(3, 30)*, **min_samples_leaf**: *randint(4, 30)*, **bootstrap**: *[True, False]*, **criterion**: *[entropy, gini]*, **class_weight**: *[balanced, None]*
KNN	**n_neighbors**: *[1, ..., 40]*
SVM	**kernel**: *[linear, poly, rbf, sigmoid]*

3.2 Decision Tree

Decision trees require little to none data preparation: the only part that must be taken care of is the presence of null values, which aren't supported. Due to the characteristics of our dataframe, there are no null values so we could simply use this method without further ado.

Table 3.2: Model selection results for Decision Tree.

Criterion	Splitter	Max Depth	Min Leaf	Min Split	Accuracy	F1_Score
gini	best	10	31	10	0.749	0.737
gini	best	4	4	19	0.748	0.734
gini	best	6	35	25	0.748	0.738
entropy	best	8	36	5	0.748	0.741
gini	best	6	32	21	0.748	0.734

Using the highlighted parameters, we generated the tree shown in Figure 3.1 that brought us to the confusion matrix 3.2 and the report 3.3.

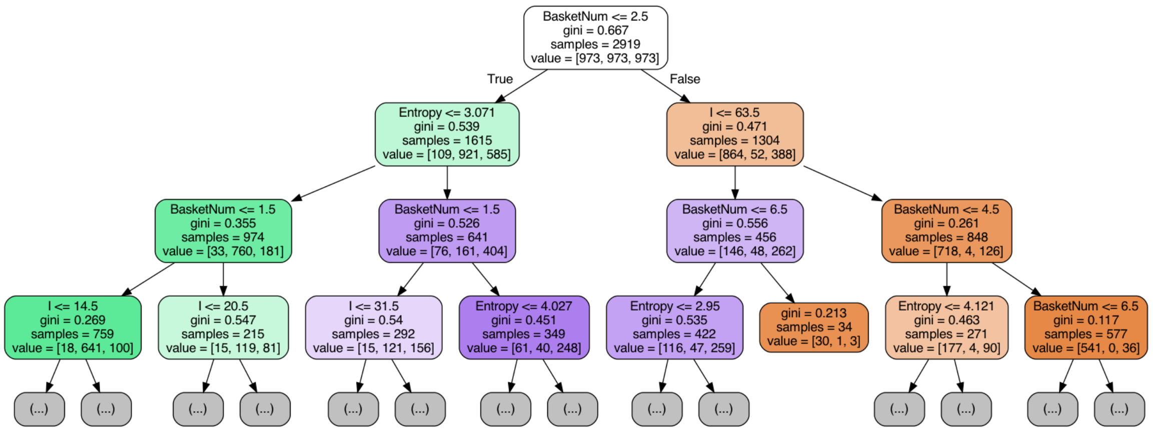
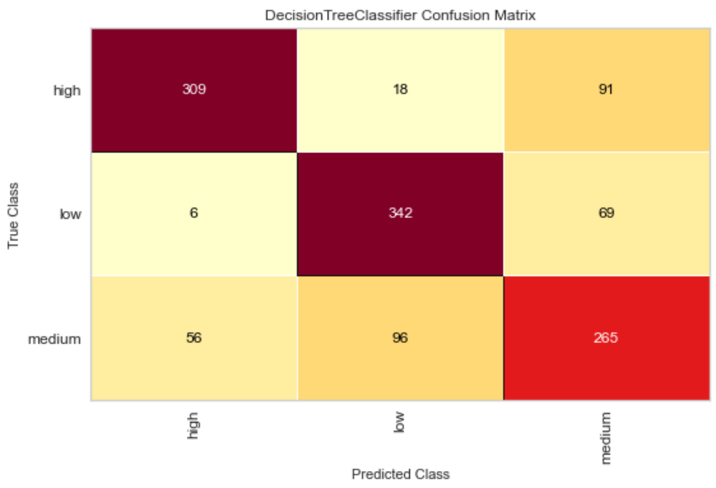
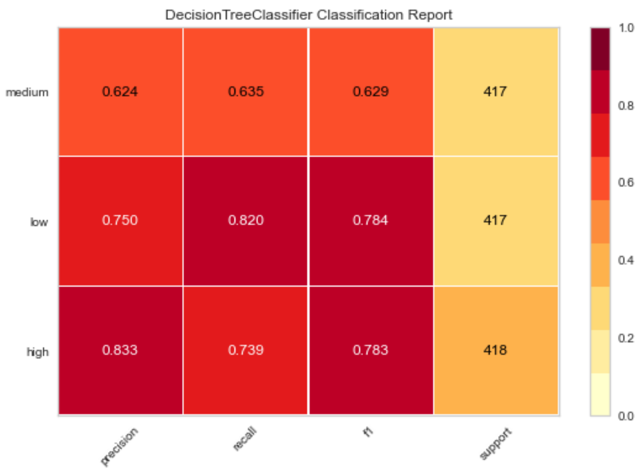


Figure 3.1: Visualization of the obtained tree.



True Class \ Predicted Class	high	low	medium
high	309	18	91
low	6	342	69
medium	56	96	265

Figure 3.2: Confusion Matrix for Decision Tree.



	precision	recall	f1	support
medium	0.624	0.635	0.629	417
low	0.750	0.820	0.784	417
high	0.833	0.739	0.783	418

Figure 3.3: Classification Report for Decision Tree.

3.3 Random Forest

Decision Trees tend to overfit on the training set. For this reason, building many of these and averaging their behavior through the well-known *wisdom of the crowd* leads to the *Random Forest* classification method. Through its randomness, we could achieve better results, as expected.

Table 3.3: Model selection results for Random Forest.

Criterion	**Bootstrap**	**Class Weight**	**n_estim**	**Max Features**	**Max Depth**	**Min Leaf**	**Min Split**	**Accuracy**	**F1_Score**
entropy	True	None	750	1	None	13	25	0.762	0.740
entropy	True	None	750	1	None	12	20	0.762	0.754
entropy	True	None	300	2	8	20	20	0.761	0.742
gini	True	Balanced	1000	1	12	10	22	0.760	0.763
entropy	True	Balanced	300	2	10	19	27	0.758	0.754

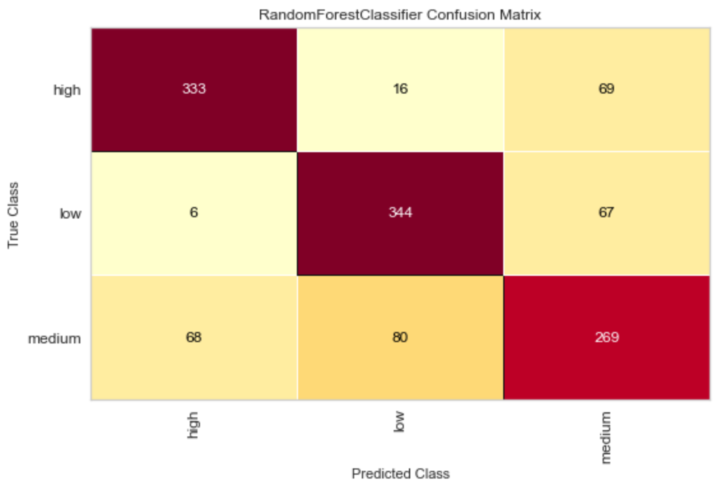


Figure 3.4: Confusion Matrix for Random Forest.

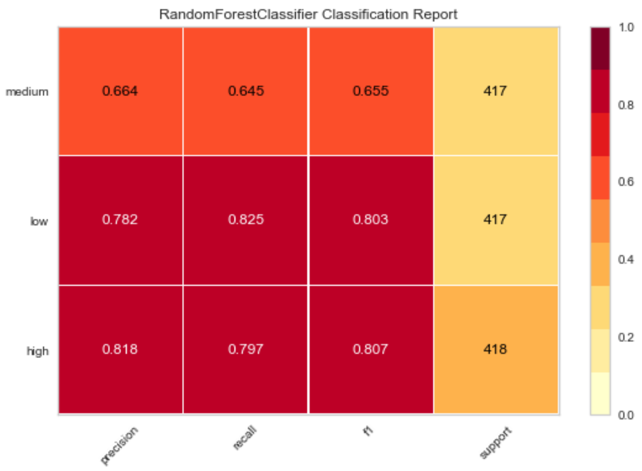


Figure 3.5: Classification Report for Random Forest.

3.4 SVM

The model selection returned Linear as the best kernel (3.4). This is easily explainable by the fact that our data can be imagined as actually divided by the three labels we defined for this task.

Table 3.4: Model selection results for SVM.

Kernel	**Accuracy**	**F1 Score**
Linear	0.729	0.728
Rbf	0.700	0.699
Sigmoid	0.586	0.579
Poly	0.569	0.547

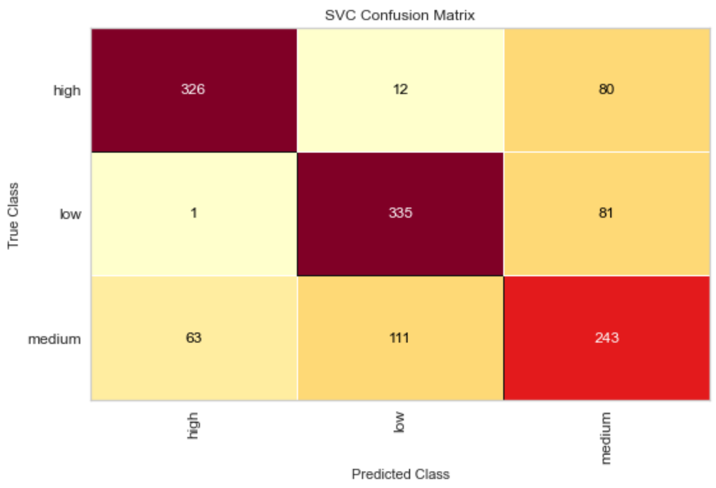


Figure 3.6: Confusion Matrix for SVM.

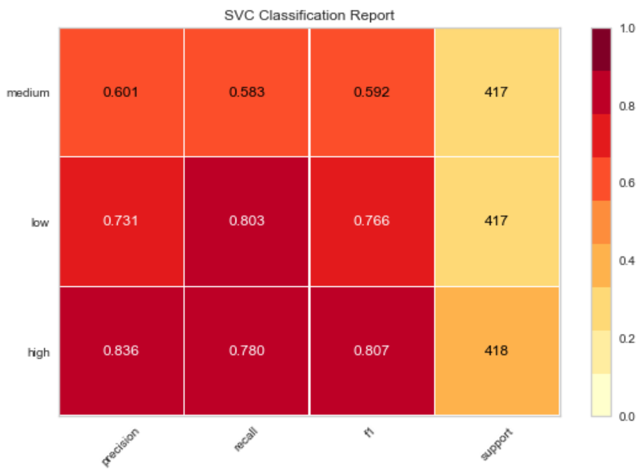


Figure 3.7: Classification Report for SVM.

3.5 KNN

Another famous learning method we thought would be interesting is the *KNN*. First of all, we had to find the best value for hyperparameter k. So, we iterated from **k = 1** to **k = 40** to see the different variations on the *error rate*, as shown in Figure 3.8. Finally, we have chosen the value which had the least error rate (**k = 12**, with **0.26% ER**) and computed again the usual results.

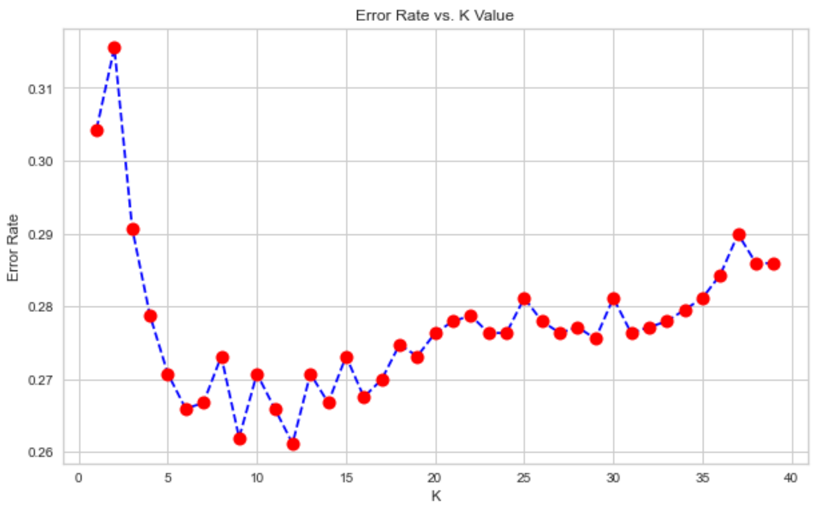


Figure 3.8: Mean of the Error Rate in KNN

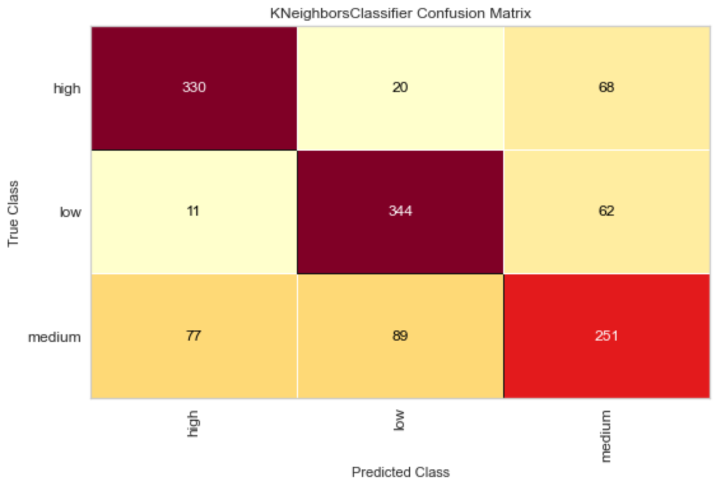


Figure 3.9: Confusion Matrix for KNN.

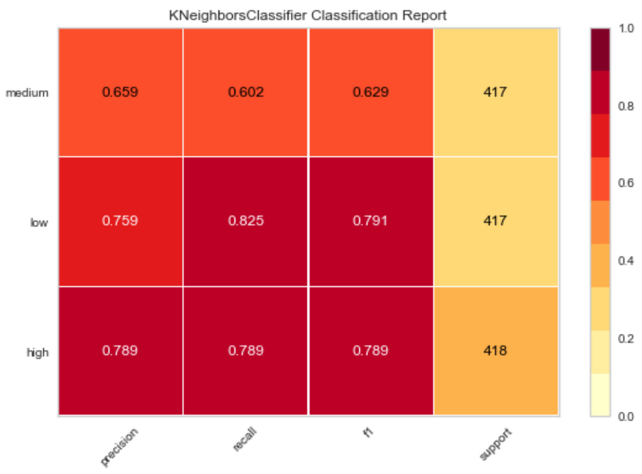


Figure 3.10: Classification Report for KNN.

3.6 Naive Bayesian

For completeness, we used the *Naive Bayesian* classifier. Its strengths are the correct handling of missing values, and robustness to both irrelevant attributes and noise data. Due to all of the pre-processing on our constructed data, these strengths aren't useful: there are no missing values, the noise has been reduced with the outliers detection and irrelevant attributes were already dropped at the beginning.

We've tried the different *Naive Bayesian* classifiers provided by `sklearn`, and the results that are shown in the following pictures 3.11 and 3.12 refer to the best one, that is the `GaussianNB`, which assumes that the likelihood of the features is *gaussian*.

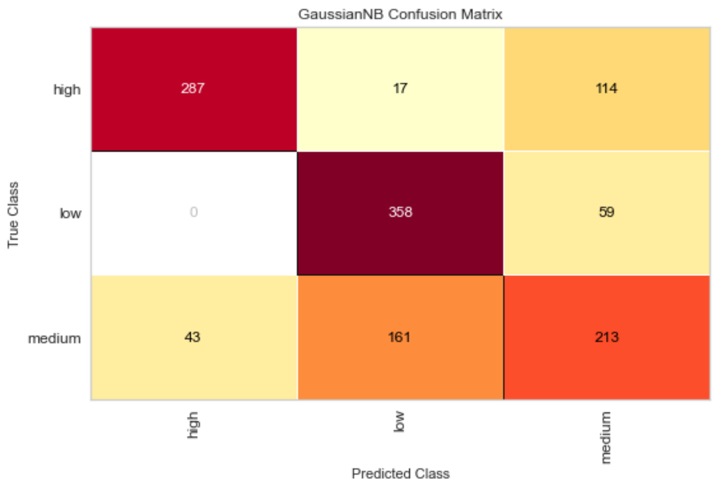


Figure 3.11: Confusion Matrix for Naive Bayesian.

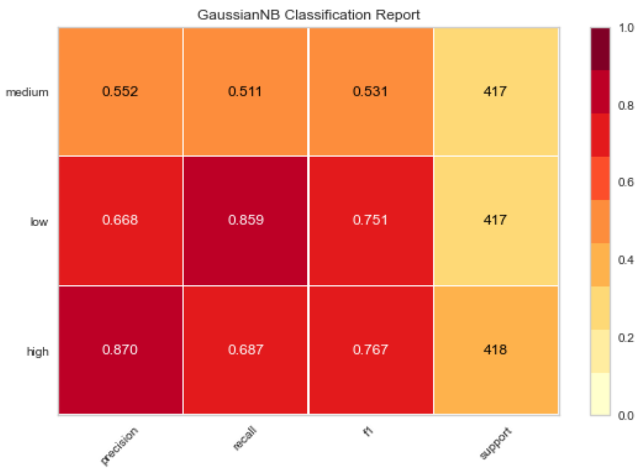


Figure 3.12: Classification Report for Naive Bayesian.

3.7 Comparison of used predictive models

We finally present a table (3.5) that summarizes the resulting timings required and accuracies on train set and test set.

3.7.1 Fit Time

SVM's slowness depends on the complexity dominating factor, which is the number of records of our training set. Considering that we've chosen the *linear* kernel, this can be reduced by a small margin by using the `LinearSVC` classifier which has a better implementation.

About *Random Forest* being the slowest out of all of the methods, this is reasoned by the parameter `n_estimators` (**=750**)

Table 3.5: Times and Accuracies for each predictive algorithm.

Algorithm	Fit Time (ms)	Prediction Time (ms)	Accuracy (Train Set)	Accuracy (Test Set)
Decision Tree	0.005	0.003	0.771	0.746
Random Forest	1.698	0.133	0.791	0.757
KNN	0.004	0.033	0.760	0.730
SVM	1.075	0.012	0.727	0.727
Naive Bayesian	0.004	0.002	0.682	0.682

in the final execution. This parameter represents the number of trees that the algorithm will produce, so higher values mean higher performance at the cost of being slower.

3.7.2 Accuracy

Let's start by analyzing the performance increase between *Decision Tree* and *Random Forest*. As we previously stated in Section 3.3, we expect that *Random Forest* corrects the *Decision Tree* overfitting behavior retrieving better results, and this is what happened. In fact, as Table 3.5 shows, *Random Forest* obtains an increase in the `Accuracy` of almost **3%** on the *train set* and **1%** on the *test set*. This increment isn't very significant, but it's interesting to look at how the features' importance changes between the two methods (Figure 3.13 and Figure 3.14).
The shown Feature Importance is called `Gini Importance`, computed as the (normalized) total reduction of the criterion brought by that feature. *Decision Tree* gives a lot of importance to the `BasketNum` attribute, but this may be biased because when looking at the *feature importance* in the *Random Forest* this inequality disappears.

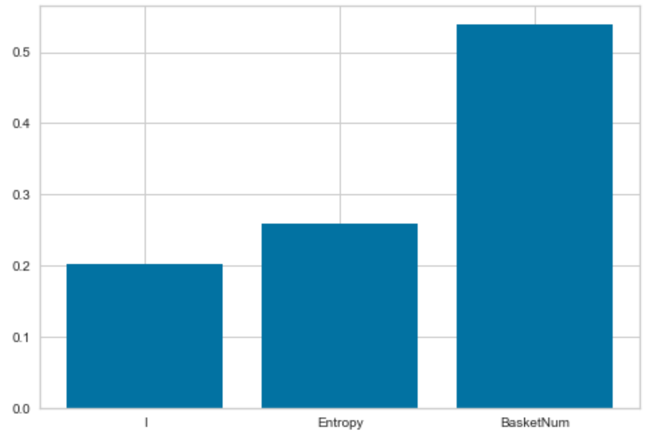


Figure 3.13: Feature Importance for Decision Tree.

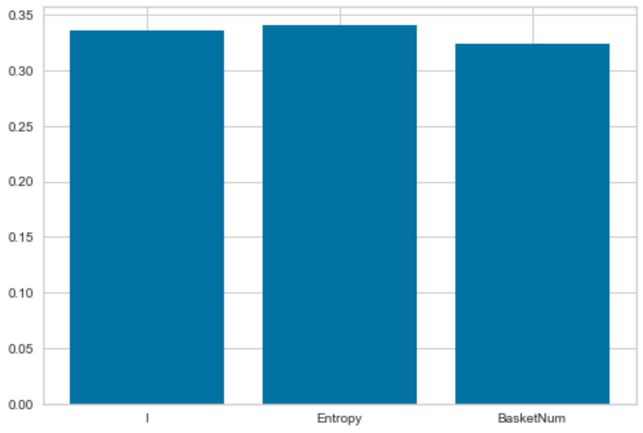


Figure 3.14: Feature Importance for Random Forest.

The remaining methods gave back similar results, depending on features of algorithms themself, like the ones described in the *Naive Bayesian* section.

3.7.3 Conclusions

By looking at results (Table 3.5) and comparing the ratio between the time required and the accuracy, we can see that the *Decision Tree* wins above all others. It resulted fast both in the fitting and prediction parts, as expected by the properties of the method, while still scoring up to second place in accuracy.
As it was previously stated, the results obtained by the *Random Forest* classification technique are better, but at one cost: time expenditure. Due to all of our data processing, data's small enough that this time cost is not significant (around 1 second and a half), but in the case of much bigger datasets this may be a problem.
The other tested techniques gave good scores anyway, just not as good as the first two: this was visually highlighted through all the classification chapter by the confusion matrix's results for each model.

Chapter 4

Sequential Pattern Mining

4.1 Sequence Generation

First of all, we needed to decide which customers had to be filtered due to computational constraints. We started from the subset of customers obtained after the outliers' detection described in Section 1.4 (**4171** customers). This initial set of customers was modeled as sequences of (**14929**) baskets that were further filtered according to their length. The boundaries of this filtering were inferred by the *mean* and *stdev* values for the `Imax` attribute (see Table 1.1). Retrieving baskets with a length between **6** and **80**, dropped down the number of baskets from **14929** to **11229**, that were used as input for the `GSP` algorithm.

4.2 GSP Results

After some experiments using the proposed implementation of the `GSP`, we noticed that computation times were too long. So we opted to use another `GSP` implementation that exploits multiprocessing [1].

We've run many executions with different support thresholds to find interesting results, but even with a support count as low as **80** (precisely, **78** or **0.7%**), we have obtained patterns with a maximum of two elements and none of three or more. Results are shown in Tables 4.1 and 4.2.

Run	**Candidates**	**Filtered**
1	3751	993
2	986049	31
3	59319	0

Table 4.1: GSP Results.

Pattern	**Support Count**
Wooden Frame Antique White, Wooden Picture Frame White Finish	138
Pink Regency Teacup and Saucer, Green Regency Teacup and Saucer	134
Gardeners Kneeling Pad Cup Of Tea, Gardeners Kneeling Pad Keep Calm	132
Alarm Clock Bakelike Red, Alarm Clock Bakelike Green	130
Paper Chain Kit 50's Christmas, Paper Chain Kit Vintage Christmas	122

Table 4.2: Patterns ordered by support count.

It immediately catches the eye all the patterns correlate variants of the same item: just looking at the second row, people buying *Pink Regency Teacup and Saucer* are more likely to buy the *Green* variant as well.

[1] https://github.com/jacksonpradolima/gsp-py

Often, items are bought also because they are part of a larger set, like in the third and fifth row, made by objects that relate to a "*tea set*" or "*Christmas decorations*".

After analyzing the whole view of our results (which is stored in the results.txt file), especially the variety of best seller items (length-1 patterns), we can safely state that the store isn't specialized in a specific products' field, but it sells many items from completely different areas: these can range from simple house decorating *(Alarm Clock)*, to outdoor experiences *(Lunch Bag)* or even children games *(Poppy's Playhouse)*.

Also, from the customer's point of view, the very low support values retrieved for the patterns suggest that clients come with different interests as well. Considering that all length-2 patterns have a support value lower than **1%**, most baskets contain really diverse items. This, and the fact that items have relatively low cost, suggests that it's a store that wants to focus its sales around gifting activities, like holiday periods *(e.g. Christmas, etc.)*.

Chapter 5

Association Rules Mining

5.1 Frequent Patterns

Before anything else, we need to compute the frequent itemsets. The definition of a *set* requires that items aren't repeated and this is already granted by default in our original dataset due to the `Qta` attribute: in every basket, there won't be any item that is repeated more than once.

Just as a reminder, there's a total of **4338** customers and **18532** total baskets that were analyzed.

We thought that any item with at least **1%** of support was good enough considering the retrieved results in Chapter 4. Moreover, we selected only sets with at least **3** items to stay away from both unimportant itemsets and a too high number of patterns.

With these parameters' configuration, we've obtained **53** frequent patterns. This result can be compared with the ones obtained for other *support* values, shown in Figure 5.1. The comparison can be done also among the different types of sets (Figures [5.1, 5.2, 5.3]), always with many *support* values. As expected, increasing the *support* value, the resulting number of itemsets decreases.

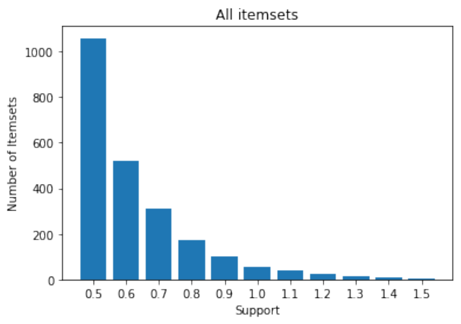


Figure 5.1: All Itemsets.

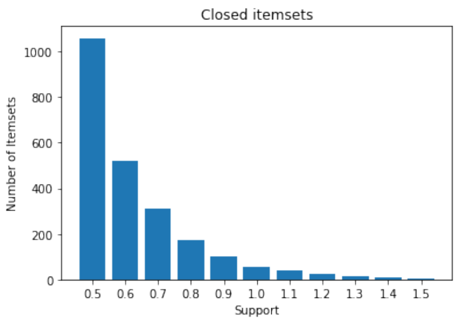


Figure 5.2: Closed Itemsets

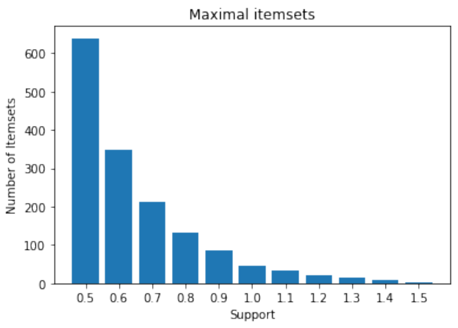


Figure 5.3: Maximal Itemset

What's interesting to note is that our results retrieved for *closed* itemsets are exactly the same for the most general case, the *all* itemsets: by definition, *maximal* is a subset of *closed* that is itself a subset of *all*, but having these last two collide means that there's no superset that has the same support out of all of them.

In Table 5.1 are listed the most significant patterns with highest supports, which are the same for all the types of itemsets.

Table 5.1: Frequent Itemsets (ordered by Support percentage).

Itemset	Support Count	Support (%)
Pink Regency Teacup and Saucer, Green Regency Teacup and Saucer, Roses Regency Teacup and Saucer	390	2.10
Green Regency Teacup and Saucer, Roses Regency Teacup and Saucer, Regency Cakestand 3 Tier	312	1.68
Lunch Bag Pink Polkadot, Lunch Bag Black Skull, Lunch Bag Red Retrospot	308	1.66
Lunch Bag Pink Polkadot, Lunch Bag Cars Blue, Lunch Bag Red Retrospot	279	1.51
Pink Regency Teacup and Saucer, Green Regency Teacup and Saucer, Regency Cakestand 3 Tier	271	1.46
Alarm Clock Bakelike Pink, Alarm Clock Bakelike Green, Alarm Clock Bakelike Red	267	1.44
Pink Regency Teacup and Saucer, Roses Regency Teacup and Saucer, Regency Cakestand 3 Tier	265	1.43
Lunch Bag Cars Blue, Lunch Bag Black Skull, Lunch Bag Red Retrospot	262	1.41
Lunch Bag Pink Polkadot, Lunch Bag Cars Blue, Lunch Bag Black Skull	260	1.40
Lunch Bag Woodland, Lunch Bag Spaceboy Design, Lunch Bag Red Retrospot	258	1.39

5.2 Association Rules

After the generation of frequent patterns, we computed the *association rules* through the *confidence* metric with a threshold of **70%**. The parameter's choice is the best trade-off between a relevant number of rules and a not too low confidence, as shown in Figure 5.4. In addition to this, from the scatter plot in Figure 5.5, our choice defines the starting point from where the *lift* value gets relevant (*from 20 up to nearly 60*).

The chosen level of *confidence* led to the results shown in Table 5.2. There are a total of **94** returned association rules that are coherent with conclusions made in the previous Chapter, about the variety of items and customers' interests.

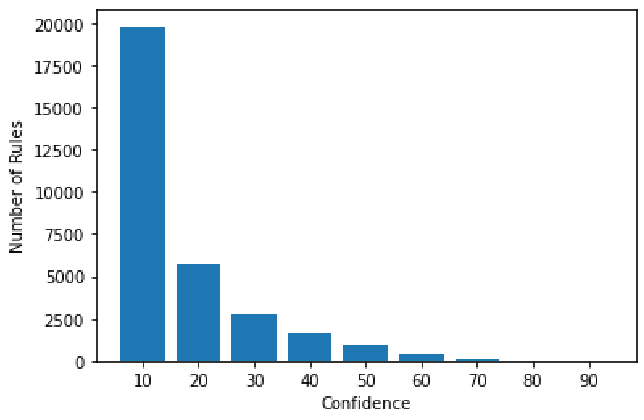


Figure 5.4: Number of rules for different Confidence values.

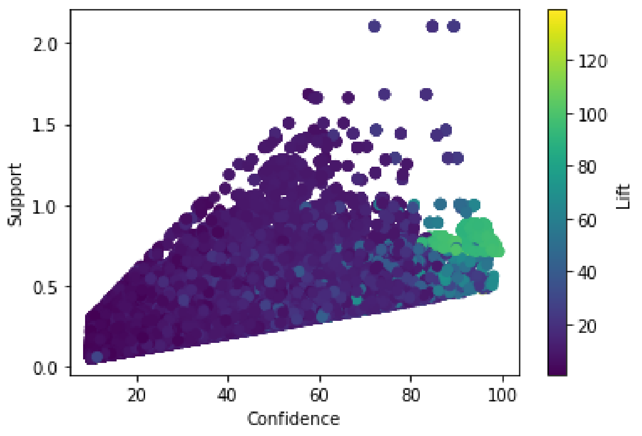


Figure 5.5: Scatter plot with Confidence, Support and Lift

Table 5.2: Association Rules (ordered by Confidence).

Antecedents	**Consequents**	**Support (%)**	**Confidence (%)**	**Lift**
Regency Tea Plate Pink, Regency Tea Plate Roses	Regency Tea Plate Green	0.99	94	64.45
Set/6 Red Spotty Paper Cups, Set/20 Red Retrospot Paper Napkins	Set/6 Red Spotty Paper Plates	0.95	92	52.56
Wooden Tree Christmas Scandinavian, Wooden Heart Christmas Scandinavian	Wooden Star Christmas Scandinavian	0.98	91	38.94
Poppy's Playhouse Livingroom. Poppy's Playhouse Bedroom	Poppy's Playhouse Kitchen	1	90	48.59
Pink Regency Teacup And Saucer, Roses Regency Teacup And Saucer	Green Regency Teacup And Saucer	2	89	23.98
Regency Milk Jug Pink, Regency Teapot Roses	Regency Sugar Bowl Green	0.89	86	59.42
Regency Sugar Bowl Green, Regency Teapot Roses	Regency Milk Jug Pink	0.89	84	57.35