University of Pisa

Department of Computer Science

Data Mining
Group 18

Supermarket Analysis

Authors:

Donato Meoli
Enrico D'Arco
Luigi Quarantiello

December, 2020

Contents

1 Data Understanding **2**
1.1 Data Semantics . . . 2
1.2 Assessing Data Quality . . . 2
1.3 Variables Transformations . . . 2
1.4 Variables Distribution . . . 3

2 Data Preparation **5**
2.1 Feature extraction . . . 5

3 Clustering **8**
3.1 Preprocessing . . . 8
3.2 K-Means . . . 8
3.3 Hierarchical clustering . . . 9
3.4 DBSCAN . . . 9

4 Predictive Analysis **11**
4.1 Support Vector Classifier . . . 11
4.2 Neural Network . . . 11
4.3 Naive Bayes Classifier . . . 12
4.4 K-Nearest Neighbors . . . 12
4.5 Decision Tree . . . 13
4.6 Random Forest . . . 13
4.7 Conclusion . . . 13

1 Data Understanding

The dataset contains *471910* entries, each of them represent a purchase related to a supermarket made by a customer over a period of two years.

1.1 Data Semantics

The dataset contains 8 attributes that correspond to:

- **BasketID** (*24627*): a 6 digit integer number uniquely assigned to each purchase; if it starts with a *C*, it indicates a cancellation (*3754*);
- **BasketDate**: the day (*from 2010/12/01 to 2011/12/09*) and time (*from 6am to 21pm*) when each purchase was placed;
- **Sale** (∼*4avg*): the unit product price, all in the same currency, probably in sterling;
- **CustomerID** (*4372 + 65073na*): a 5 digit integer number uniquely assigned to each customer;
- **CustomerCountry** (*37*): the name of the country where each customer resides;
- **ProdID** (*3953*): a 5 digit + 1 letter identifier uniquely assigned to each distinct product; identical codes with different letters identify the same products with different characteristics (*e.g.*, *84997D*: '*pink piece polkadot cutlery set* vs. *84997C*: '*blue piece polkadot cutlery set*);
- **ProdDescr** (*4097 + 753na*): the description of the product purchased;
- **Qta** (∼*11avg*): the purchased quantities of each product per order.

1.2 Assessing Data Quality

In order to assess the quality of data, we proceed by removing the *5232* duplicate entries which represents the 1.11% of the entire dataset, so we will work with the remaining *466678* rows.

Then, we proceed by removing the entries corresponding to the *65073* null *CustomerID* values, the 13.94% of the dataset, since our main goal is to outline the customer behavior and there is no way to integrate them to trace the customer's orders.

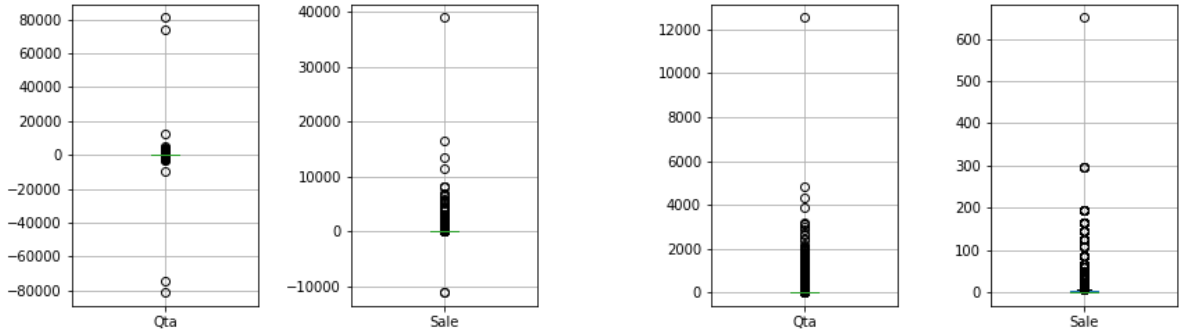
Afterwards, we removed the *ProdID* that does not respect the defined format and we found some that contains only letters, *e.g.*, '*POST*', '*D*', '*C2*', '*M*', '*BANK CHARGES*', etc. with the following respective *ProdDescr*: '*POSTAGE*', '*Discount*', '*CARRIAGE*', '*Manual*', '*Bank Charges*', etc.. As a result,we dropped *1273* entries.

1.3 Variables Transformations

In order to manage the presence of negative quantities as shown in the first boxplot in Figure 1a, we decide to keep track of the portion of each order that has been canceled. At this point we should face three different cases:

- a *cancelation exists with a counterpart*, i.e., exists an order with the same (but positive) quantity and a previous date, so both are canceled;
- a *cancelation exists without a counterpart*, this is probably due to the fact that the orders were performed before December 2010 (the entry point of the database), so we remove this;
- a *cancelation exists with multiple counterparts*, so we delete the most recent.

To also manage the presence of the remaining prices equal to zero, 34 entries to be exact, as shown in the second boxplot in Figure 1a, we filled these values with the average of the selling prices of the products with the same id.

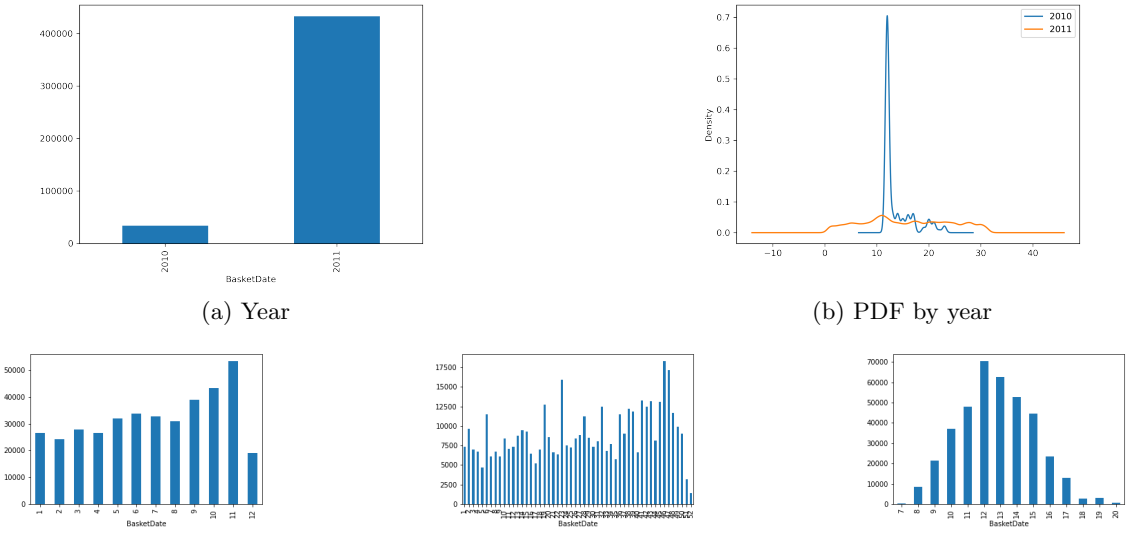


(a) (b)

Figure 1: *Qta* and *Sale* boxplots before and after data cleaning

We decided to create the attribute *TotSale*, i.e., the product between *Sale* and *Qta*, that represents the total amount spent by a customer for each type of product purchased. We made this choice mainly to have a clearer look to the dataset, emphasizing an important information that was not explicit in the original table, and also for the feature extraction later.

1.4 Variables Distribution



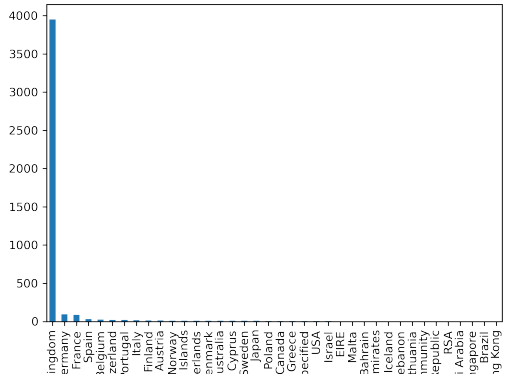
(a) Year (b) PDF by year

(c) PDF by month (d) PDF by week (e) PDF by hour

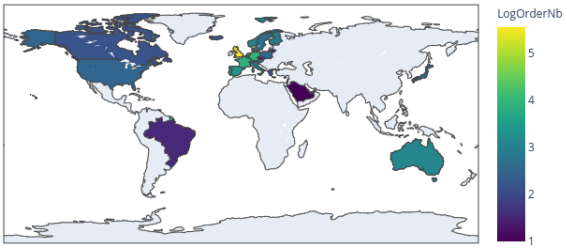
Figure 2: BasketDate distributions

From the Figure 2a, we can see that the attribute is highly unbalanced; in fact, we have that almost all the records are related to transaction of the 2011, while the objects from 2010 are very few. Indeed, the rows of 2011 represent about the 93% of the whole dataset.
Furthermore, from the Figure 2b, that represents an estimation of the probability density function of *BasketDate* divided by year, we can appreciate the two different distributions.
In fact, for the 2010, we have a very uneven plot, which indicates that the records are not uniformly distributed

with respect to the days in a month. This because, for the majority of the months in 2010, there were registered only transactions from a single day, the 12th; this is the value for which the plot shows the peak. On the other hand, the distribution for the 2011 is much more homogeneous, meaning that the transactions were registered for most days in the months of that year. For these reasons we decided to remove the entries of the 2010, since the data were not taken regularly, and so they could alter our next study. We will focus on the 2011 entries, which were present much more uniformly and for this reason they could be more representative.
Other interesting distributions are plotted in Figure 2c, 2d and 2e. In the first one, we can see that the last weeks of the year are the one with more purchases; that is consistent with our expectations, since those are the weeks closest to Christmas time, that typically represents a great period of shopping. This thesis is supported also from the second one, in which we can see that December is the month with the most purchases. The third one is focused instead on the hours in a day; we found that, unsurprisingly, the most popular hourly is lunchtime.



(a)



(b)

Figure 3: CustomerCountry distributions

In Figure 3a and 3b, we can see the distribution of the *CustomerID* with respect to the country; from the plot, it is clear that the most frequent country is the *United Kingdom*, that is present in about the 90% of the rows.

Finally, we see some informations about the correlation of the attributes, to see if some of them are redundant. From the Figure 4, we can see that almost all the attributes are uncorrelated, except for *TotSale*, that shows ah high correlation with *Qta*; that follows what we expected, since *TotSale* is, by construction, dependent on *Qta*. So, we conclude that all the original columns are independent, and so we don't need to perform any further manipulation.

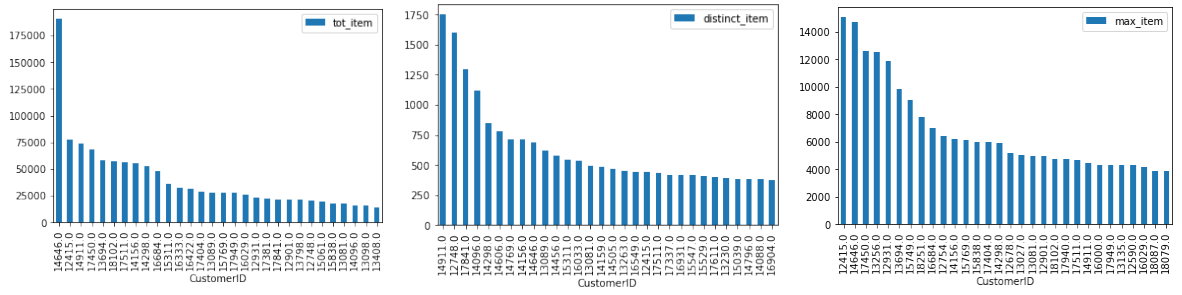
Figure 4: Correlation Matrix

We ended up with a cleaned dataset, consisting of *363577* entries.

2 Data Preparation

We also looked at the outliers we already detected with the box plots in the previous section.
As we can see in Figure 1b, the previous operations already cleaned some of the outliers we had in the original dataset; now we manually check them, to determine if they are errors or not.

In the case of **Sale**, we have that the maximum value is 649.5, which is quite high; in reality, we discovered that all the outliers for this attribute are referring to purchases of furniture or objects sold in big quantities, *e.g.* 60 picnic baskets. So we consider these rows as correct.
In the case of **Qta**, we see just two values that are really away from the others, and they represent huge purchases, with quantities equal to 74251 and 80995; we have also the same quantities in negative. Since this is a very strange case, we thought that they could be the results of an error and so we decided to drop these 4 rows.



(a) Total number of items per customer

(b) Number of distinct items per customer

(c) Maximum number of items per customer

Figure 5: Visualization for the extracted features

2.1 Feature extraction

Here, we extract new features from the dataset, in order to describe the customers behavior.
First of all, we defined some basic features, like:

- The total number of items purchased by a customer
- The number of distinct items bought by a customer
- The maximum number of items purchased by a customer during a shopping session

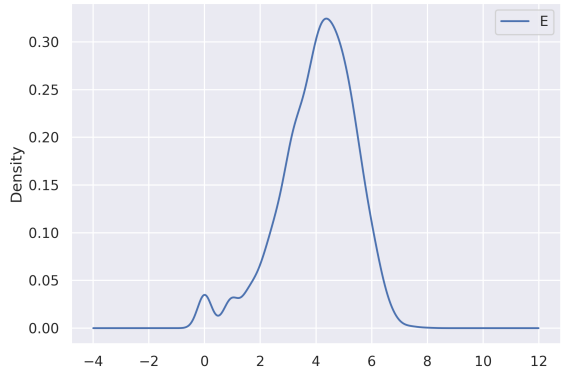


Figure 6: Entropy distribution

In Figure 5, we can see some visualization for these features; in particular, they represent the first 30 customers with the biggest values for each feature.
An interesting information is clear from the plot 5c, where we can see that the maximum quantities purchased in a single shopping session are very big; they are all above 3500, with the maximum equal to 15049. These are very high values, unlikely for a retail customer; this led us to think that the supermarket in question also sells wholesale.

In Figure 6, we can evaluate the distribution of the entropy. In particular, we are focusing on the attribute **TotalPrice**. This value of the entropy represents the variability of the customer's spending habits; a bigger value means that the customer did not have a regular

behavior, instead he spent always different amount of money.
We have a small peak corresponding to 0, meaning that there are some customers with very specific habits, but the maximum is reached for ~ 4, which means that the majority of the clients have a quite unpredictable behavior.

Since the focus is on the purchasing behavior of the customers, we decided to study other features, in order to deeply understand the information we have. We chose to extract information about:

- The number of basket per customer
- The number of *bad* basket per customer
- The number of purchased products
- The number of returned products
- The amount of money spent for each customer
- The amount of money refunded per customer

Thanks to these features, we were able to outline better the kind of customers registered in the dataset.
First, we checked if there were customers with a number of returned products bigger than the purchased ones; as a matter of fact, we found **18** rows of this kind. We consider these as *bad* customers, since they have made more returns than purchases; this is clearly an impossible situation, maybe due to the fact that some transaction was not registered in the dataset. Anyway, we decided to delete these customers.
Plus, we inserted two other columns:

- **ActualQta**, which is the algebraic sum between the positive and the negative quantities
- **ActualSpent**, equal to the difference between the money spent and the one refunded

We also deleted all the customers with the **ActualQta** equal to zero, since, in the end, they didn't buy anything.

Now we extract some information about the average values, as:

- **AvgPrice**, the ratio between **ActualSpent** and **ActualQta**, that represents the average price of the products bought by a customer
- **AvgBaskValue**, the ratio between **ActualSpent** and the number of *good* baskets, which represents the average amount of money spent for each basket per customer

Another interesting feature is the **MonthFreq**, that we computed as the ratio between the number of *good* baskets and 12, the number of months. We can visualize this attribute in Figure 7, where we can clearly see that the peak is really close to 0; in fact, the mode of this column in equal to **0.083**, which means that the majority of the customers went to this supermarket just once in the year.

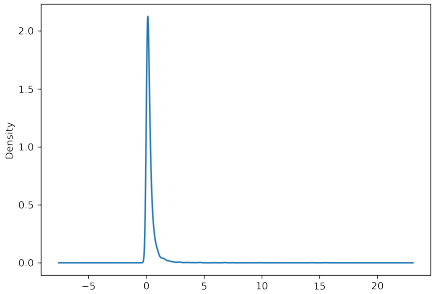
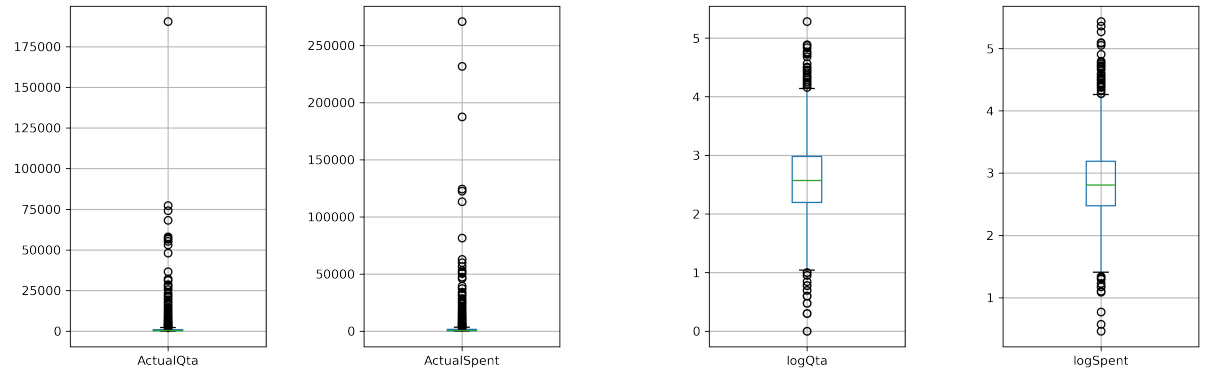


Figure 7: Monthly Frequency

Analyzing the Figure 8a, we can see that the attributes **ActualQta** and **ActualSpent** are really spread; for this reason, we consider their logarithm in base 10, so that the difference between high values has a different weight. We can appreciate the changes in the Figure 8b, where the two distributions are much more compact.
After the transformations of these attributes, we saw that also **AvgPrice** and **AvgBaskValue** were quite spread, and so we decided to compute the same operation to them.



(a) (b)

Figure 8: Box plots for ActualQta and ActualSpent, before and after the log

Now, we can visualize the correlation between the attributes. In Figure 9, as expected, we find that the average basket value is highly correlated with the total quantity purchased and the amount of money spent. Furthermore, we can see that also the monthly frequency is correlated with the amount spent and the total quantity. In the end, of course, the quantity purchased is very highly correlated with the total amount spent.

In Figure 10, we can visualize the pairplot for the attributes we have.
On the diagonal, we have the density plots, where we can appreciate the distribution of the features. Instead, the other scatter plots show the relationship between two variables. By analyzing those, we can have a confirmation of what we already found thanks to the previous plot.

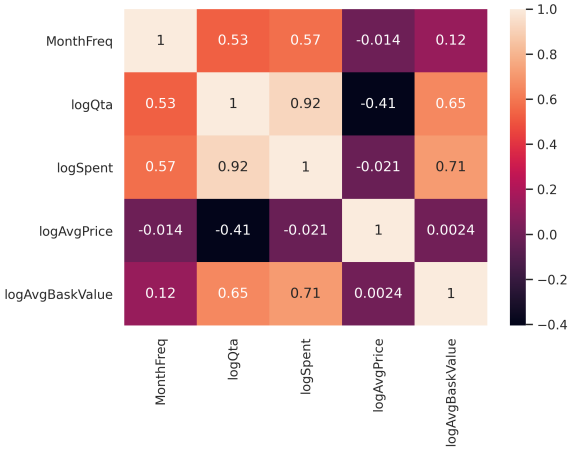


Figure 9: Correlation Matrix

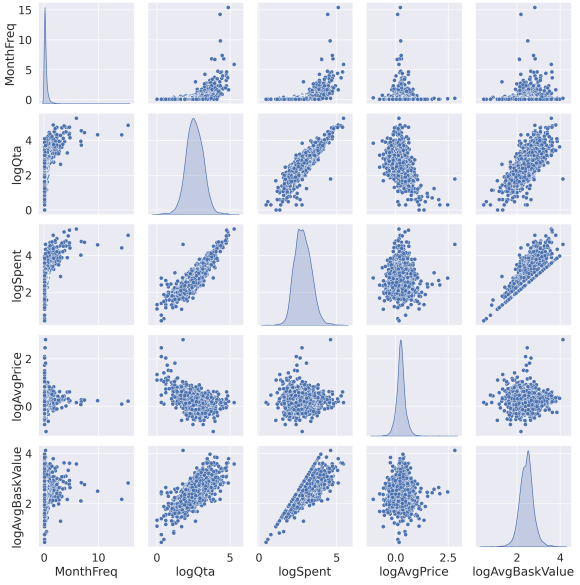


Figure 10: Pairplot

3 Clustering

We now run and compare some clustering algorithm in order to find some structures among the data. First we start with a basic *K-Means* followed with some *Hierarchical clustering techniques* and *DBSCAN*.

3.1 Preprocessing

The data matrix was first standardized and then the two principal components were extracted. The choice was between two and tree principal component since they retain respectively ∼0.75 and ∼0.85 of the variance of the data.

We selected the two principal components because it performed better on K-Means and, in this way, we could also have a visual inspection.

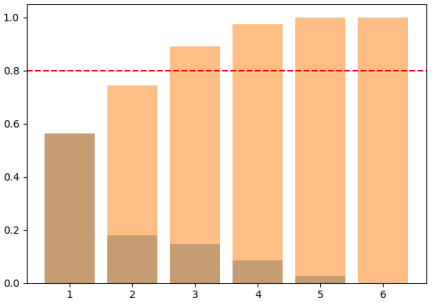
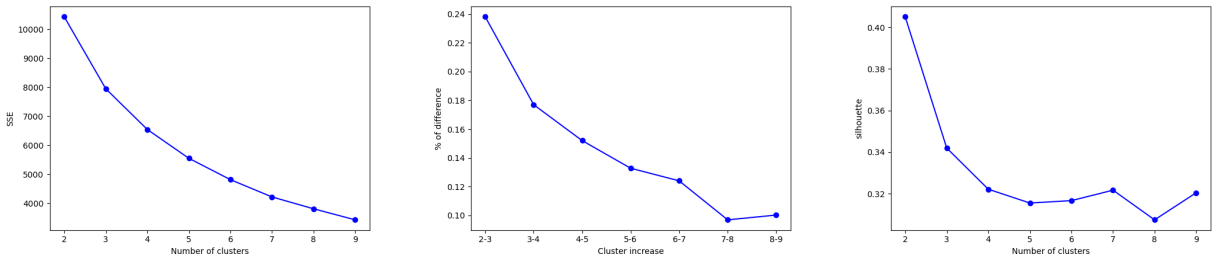


Figure 11: Explained variance ratio

3.2 K-Means

The algorithm run for K ranging from 2 to 10 clusters. For each iteration the SSE and the average silhouette value were computed.



(a) SSE (b) % of decrease in SSE (c) Average silhouette

Figure 12: *K-Means* metrics

By looking at the plots in Figure 12, we can see that there isn't a prominent *elbow shape*, so we needed the analysis of more metrics to get the optimal number of clusters. At first, we saw that, for K moving from 2 to 3, the difference of the SSE in percentage was quite high (∼0.24), while for all the other values the difference was below 0.2. By inspecting the silhouette score, it is possible to see that it has its maximum for $K = 2$, followed by $K = 3$. In conclusion, we opted for 3 clusters, to get a trade off between high silhouette and low SSE.

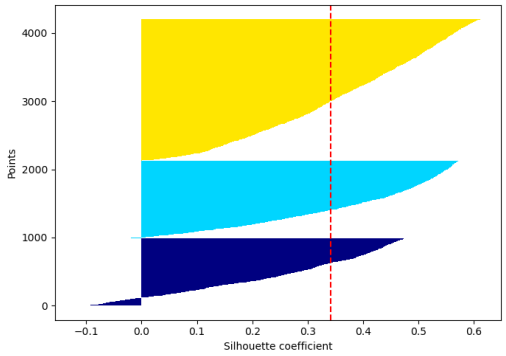
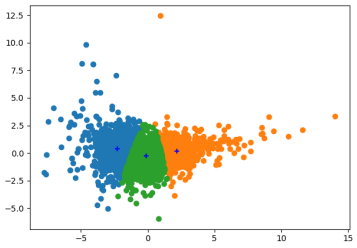


Figure 13: Silhouette score for each data point

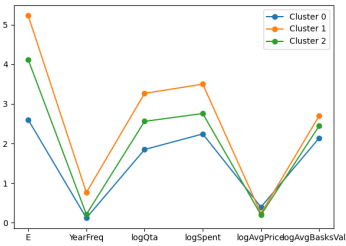
The resulting clusters have a comparable number of points. In particular *Cluster 0* has **990** points, *Cluster 1* **1137** and *Cluster 2* **2072**.
In Figure 13, we can see the plot of the Silhouette score, where each different color represents a different cluster, and the dotted red line is the average silhouette score. We see that a small fraction of points in *Cluster 1* and *Cluster 0* have a negative value but overall the Silhouette suggests a good clustering.

In Figure 14a, we can appreciate the results of the algorithm, where we can see that the clusters are well separated from one another.
The similarity matrix (Figure 14b) shows the affinity of elements



(a) Clustering result

(b) Similarity heatmap



(c) Average values per cluster

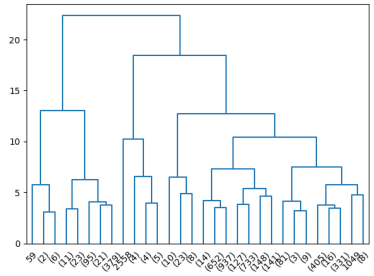
inside a cluster; we can see that the results are pretty good, and in particular *Cluster 2* contains points really similar to each other.
Instead, the plot in Figure 14c shows the average values for each attribute divided by the clusters. In particular, it is possible to see that *Cluster 0* contains the most frequent and spending customers, followed by *Cluster 2* and *Cluster 1*.

3.3 Hierarchical clustering

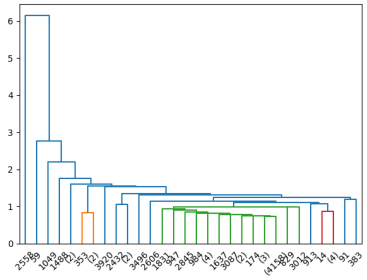
We used different kinds of algorithm: *Complete Link*, *Single Link*, *Ward* and *Centroid*. The CPCC coefficient are respectively 0.51, 0.62, 0.44 and 0.74 meaning that the best results are obtained with *Single Link* and *Centroid*. By looking at the dendograms in Figure 15, if we cut the tree by selecting two clusters, we can see that the vast majority of the data falls into a single cluster, leaving the last merge between a large cluster and a small one, in some cases even a singleton. The most balanced result is obtained with the *Ward* method, even though it has the lowest CPCC.

3.4 DBSCAN

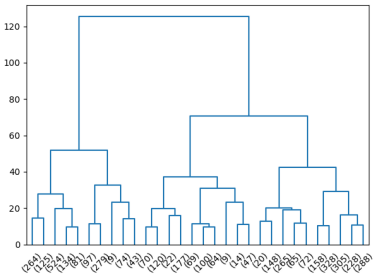
We explored different combination of *eps* and *MinPts*; for a given value of *MinPts*, we selected *eps* by checking the *KNN* distance, with K equal to *MinPts*.
Some results are plotted in Figure 16. In the first case, with *eps*= 0.6 and *MinPts*= 8, we have a large and cluster surrounded by noise points; in the second one, with *eps*= 0.3 and *MinPts*= 5, we still have a large cluster, but also a lot of other clusters, really small if compared to the central one.
For that, we think that the dataset is not suitable for this kind of algorithm.



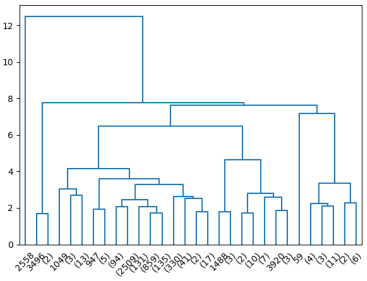
(a) *Complete Link*



(b) *Single Link*

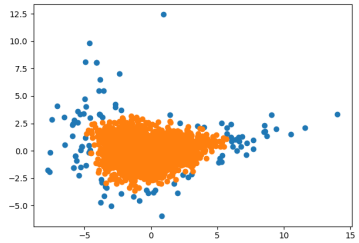


(c) *Ward*



(d) *Centroids*

Figure 15: Dendograms



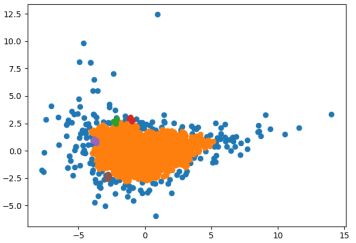


Figure 16: Results of DBSCAN

4 Predictive Analysis

In this section, we consider the problem of predicting the spending behavior of each customer.We will use the main classifier models and evaluate their performances.

First, we will use the customer profile deriving from the K-means clustering, from the previous section. From the Figure 14a, we recall that we partitioned the dataset in 3 clusters, that represent the **high-spending** customers, the **medium-spending** customers and the **low-spending** customers. We can see that the clusters are well separated from each other, and so we can use them as customer classification.

To perform the task, which is a *supervised* one, we split the dataset into training and test set, equal to the 75% and 25% of the dataset respectively. Plus, during the splitting, we specify that each partition must have approximately the same relative class frequencies; that is to manage the unbalance we have in the original dataset, in which the **medium-spending** customers are much more frequent than the other classes.
We also standardize the data with a standard scaler, achieving a distribution with 0 mean and unit variance.

For each model, we perform a grid search, to find the best values for some hyperparameters; for that, we used a *K-fold cross validation*, with $K = 5$.

4.1 Support Vector Classifier

The first model we analyze is the **SVM**, that is a classifier that searches for an hyperplane that can linearly separate the data points, possibly in a transformed space, in the non-separable case. We use a Linear Support Vector Classifier and, after the grid search, we found that the best value for the regularization parameter is $C = 1000$.
With this configuration, we achieved a training accuracy of 0.9906 and a test accuracy of 0.9962.
From the Figure 17, we can see the confusion matrix related to the results on the test set, and we notice that the model made only 4 mistakes on the whole dataset; for that, we have that also the precision and the recall for each class are all above 0.98.

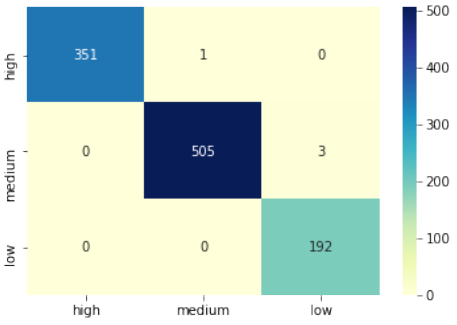


Figure 17: Confusion Matrix for SVM Classifier

4.2 Neural Network

Now, we use a Neural Network, in particular a **Multi Layer Perceptron**.
The best values for the hyperparameters that we found was:

- 1 hidden layer with 50 units
- Logistic activation function
- Learning rate equal to 0.001
- The *lbfgs* optimizer, that can converge faster and with better results on small datasets like ours

With this model, we got a training accuracy of 1 and a test accuracy of 0.999; the reason is clear in Figure 18, where we can see that just one data point was misclassified.

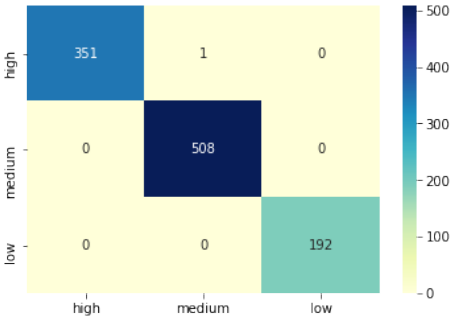


Figure 18: Confusion Matrix for Neural Network

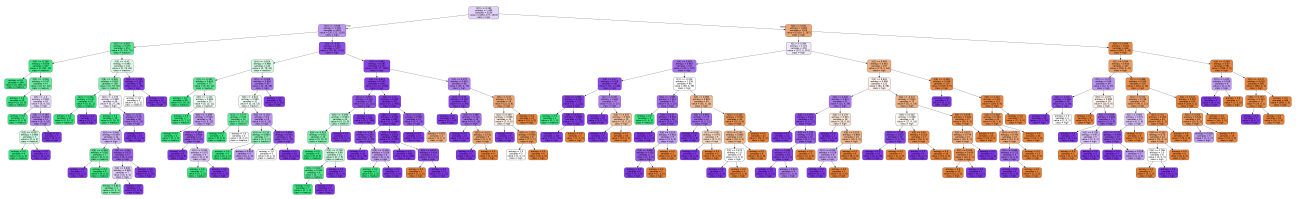


Figure 21: Decision Tree

4.3 Naive Bayes Classifier

The Naive Bayes Classifier tries to estimate the class conditional probability for each item, using the *Bayes* theorem, with the assumption that all the features are conditionally independent. In particular, we used a **GaussianNB**, where the likelihood of the features is assumed to be Gaussian.
In Figure 19, we can see that the performances of this model are slightly worse than the previous ones; in fact, we achieve a training accuracy of 0.9668 and a test accuracy of 0.9525.

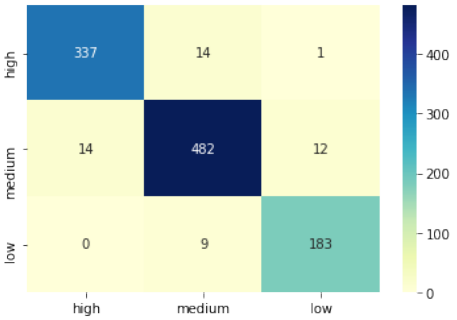


Figure 19: Confusion Matrix for Naive Bayes

4.4 K-Nearest Neighbors

The K-Nearest Neighbors model is an instance-based classifier, that, for each record, selects the class label based on the majority of its nearest neighbors.
The grid search showed that the best configuration is

- *n_neighbors* equal to 50
- *distance* as weight function; that makes the weight of a point to be inversely proportional to its distance

With these values, we have a training accuracy of 1 and a test accuracy of 0.9781. The Figure 20 shows that some points are misclassified, with some customers, belonging to the high and low profile, incorrectly labeled as medium.

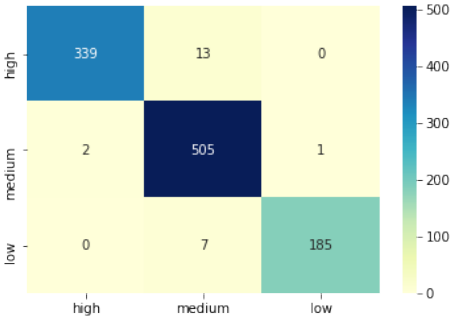


Figure 20: Confusion Matrix for K-Nearest Neighbors

4.5 Decision Tree

A Decision Tree model learns some decision rules from the training data, in order to grow a tree that is able to predict the class label for our test points. The best configuration found is:

- *criterion* equal to *gini*
- *max_depth* equal to 200
- *max_features* equal to 5
- *min_samples_leaf* equal to 1
- *min_samples_split* equal to 2
- *splitter* equal to *best*

The best result is a training accuracy of 1 and a test accuracy of 0.9392, but one of the advantages of this kind of model is that it is easy to interpret, since it can be visualized. In Figure 21, we can have a representation of the best estimator that we found.
The Figure 22 shows the model makes several mistakes, with respect to the other classifiers, given that the decision tree is a quite simple model.

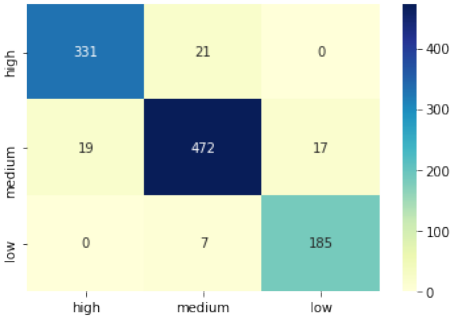


Figure 22: Confusion Matrix for Decision Tree

4.6 Random Forest

Lastly, we analyze the Random Forest. It is an *ensemble* model, composed by several decision trees, and the result is given by averaging the predictions of the single classifier. In this case, the best hyperparameters are:

- *max_depth* equal to 30
- *max_features* equal to 1
- *min_samples_leaf* equal to 1
- *min_samples_split* equal to 2
- *n_estimators* equal to 1500

As a results, we have a training accuracy of 1 and a test accuracy of 0.9952; these good results are confirmed by the Figure 23, that shows that the errors made by the model are very few.

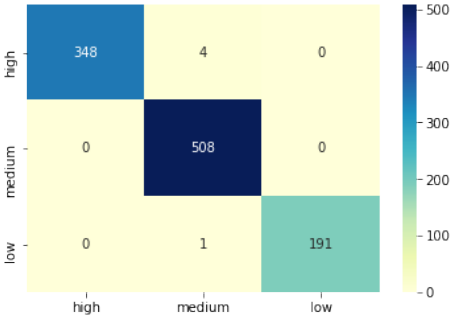


Figure 23: Confusion Matrix for Random Forest

4.7 Conclusion

In conclusion, the model that performs the best is the Neural Network, which is almost perfect both on training and test set; also the SVM and Random Forest have very good performances, having an accuracy of 99%.
Overall, even if the other models give slightly worse results, they all have an accuracy greater than 93%, which is pretty good. That is because the dataset is not very complex, since the clusters given by K-Means(Fig. 14a) are almost linearly separable.