Carvana: Don't Get Kicked!
a Kaggle Competition

Italo Guerrieri, Matteo Bogo

January 8, 2017

About the problem. Don't get Kicked! is a prediction problem submitted by Carvana to Kaggle[1]. The purpose of this competition is to create a model to predict whether if a vehicle purchased by a car dealership from an auction is in good conditions or not. Vehicles with mechanical or electrical problems that are not noticed at the time of purchase, are also referred as *Kicks*.

Introduction. This Report is an analysis of the data set supplied by Carvana with Data Mining algorithms and techniques. In the first section, the data has been explored, trying to understand variables semantic and making some plots about distribution and statistic in correlation with prediction target. After the assessment of the quality of data (redundant variables, missing values, outliers and ambiguities), and the making of opportune transformations, is possible to prepare data for further evaluations. In the second section, similarities between attributes have been measured with clustering analysis such as: partition clustering with K-Means, density-based clustering with DBSCAN and hierarchical clustering with agglomerative methods. In the third section, the extraction of frequent patterns and association rules have been made with Apriori algorithm. In the last section, classification analysis has been perpetrated with Decision Tree and Random Forest classifiers in order to obtain a real prediction for the problem.

Data Understanding

Data Semantics. This is our interpretation of the variables of the data set.

- *RefId* : a unique identifier of a record.
- *IsBadBuy* : the variable to be predicted, it is a boolean value that is equal to true if the car is a bad buy, false otherwise.
- *PurchDate* : the date on which the vehicle was bought at auction.
- *Auction* : the auction house where the vehicle was purchased.
- *VehYear* : the year in which the vehicle was built by its manufacturer.
- *VehicleAge* : the vehicle age in years at the time of purchase.
- *Make* : the vehicle manufacturer company (i.e. Audi).
- *Model* : the vehicle model (i.e. A3).
- *Trim* : the vehicle trim refers to the items that can be added to the exterior, and interior, of an automobile to increase its appeal. (i.e. s-line package)
- *SubModel* : the distinct part of a vehicle model. (i.e. Sedan)
- *Color* : the vehicle color.
- *Transmission* : the mechanism by which power is transmitted from an engine to the axle in vehicle. (Automatic or Manual)
- *WheelTypeID* : the identifier of a wheel type.
- *WheelType* : the type of wheel.
- *VehOdo* : the vehicle odometer which measures the distance travelled by a vehicle.
- *Nationality* : the nationality of the vehicle manufacturer company.
- *Size* : the vehicle segment (i.e. SUV)
- *TopThreeAmericanName* : if the vehicle is included in one of the three main US automobile company. This dataset considers as major US automobile companies: Chrysler, Ford and General Motors.
- *MMRAcquisitionAuctionAveragePrice*: estimation of the price, if sold at the auction, of the car which has a condition in the average, at the time of the purchase by the auction house.
- *MMRAcquisitionAuctionCleanPrice*: estimation of the price, if sold at the auction, of the car which has a condition above the average, at the time of the purchase by the auction house.
- *MMRAcquisitionRetailAveragePrice*: estimation of the price, if sold in a retail market, of the car which has a condition in the average, at the time of the purchase by the auction house.
- *MMRAcquisitonRetailCleanPrice*: estimation of the price, if sold in a retail market, of the car which has a condition above the average, at the time of the purchase by the auction house.

[1] for further details: `https://www.kaggle.com/c/DontGetKicked`

- *MMRCurrentAuctionAveragePrice*: estimation of the price, if sold at the auction, of the car which has a condition in the average, at the time of the sell by the auction house.
- *MMRCurrentAuctionCleanPrice*: estimation of the price, if sold at the auction, of the car which has a condition above the average, at the time of the sell by the auction house.
- *MMRCurrentRetailAveragePrice*: estimation of the price, if sold in a retail market, of the car which has a condition in the average, at the time of the sell by the auction house.
- *MMRCurrentRetailCleanPrice*: estimation of the price, if sold in a retail market, of the car which has a condition above the average, at the time of the sell by the auction house.
- *PRIMEUNIT* : if the vehicle would have an higher demand than a standard purchase.
- *AUCGUART* : the guarantee level assigned by auction house to the vehicle. (i.e. Green = Good, Red = Bad, Yellow = Warning).
- *BYRNO* : the unique number assigned by the auction house to the buyer that purchased the vehicle.
- *VNZIP1* : the zip code that identifies the city where the vehicle was purchased.
- *VNST* : the ISO 3166 code that identifies the US State where the vehicle was purchased.
- *VehBCost* : acquisition cost of the vehicle at the time of the purchase.
- *IsOnlineSale* : if the vehicle was purchased online
- *WarrantyCost* : the price of the guarantee offered by auction house.

A note about *AcquisitionType* and *KickDate*: this attributes have been provided by Kaggle in Data Dictionary, but they are not included inside the data set.

Data Statistics and Distribution. The first variable we have taken in consideration is the *IsBadBuy*. In the data set, about 87% of the records are Good Buy and 12% are Bad Buy. In Classification Analysis, this displacement will be a crucial point for our testing environment.

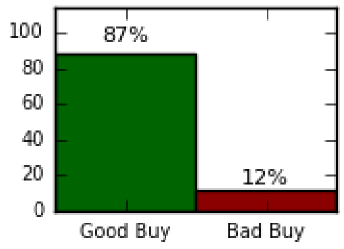


Figure 1: IsBadBuy Distribution

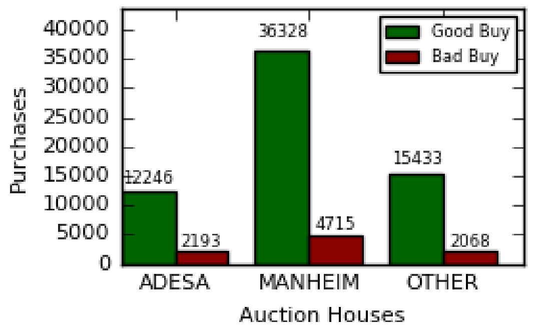


Figure 2: IsBadBuy distribution over Auction Houses

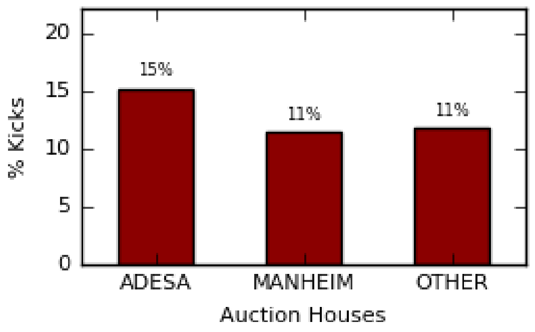


Figure 3: % of kicks over Auction Houses

Figure 2 shows the distribution of the variable *Auction*: most of cars come from the Manheim auction house, but from Figure 3, the worst in terms of *Kicks* is Adesa.

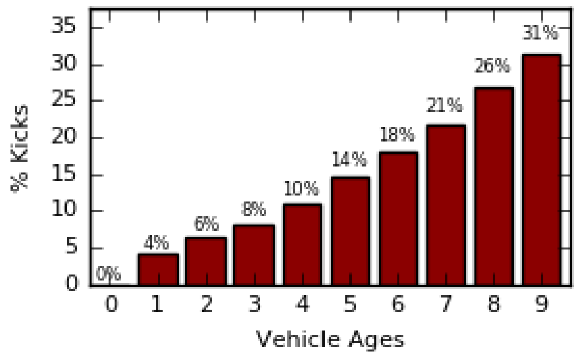


Figure 4: % of kicks over Vehicle Ages

In Figure 5 there is a distribution of the *VehicleAge* variable in function to *IsBadBuy* attribute. The diffusion of this variable follows a Gaussian distribution. In Figure 4 has to be noticed that older is the vehicle, more the probability of being a bad buy increase. Later on, this fact will be essential for building our prediction model.

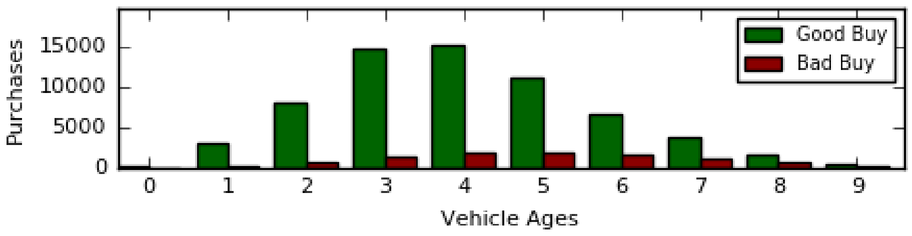


Figure 5: IsBadBuy distribution over Vehicle Ages

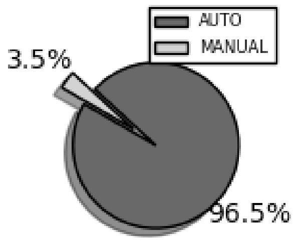


Figure 6: Transmission distribution

From Figure 6 we have observed that 96.5% of the vehicles have an automatic transmission. This seems correct due to the fact that in American culture manual transmissions are reserved only to sport cars.

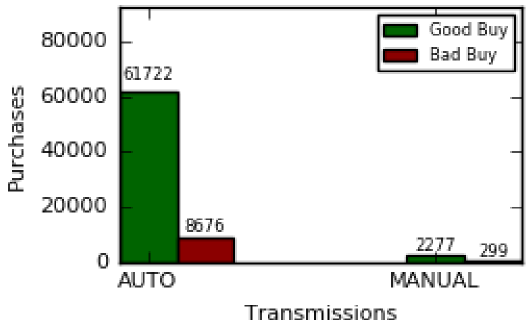


Figure 7: IsBadBuy distribution over Transmissions

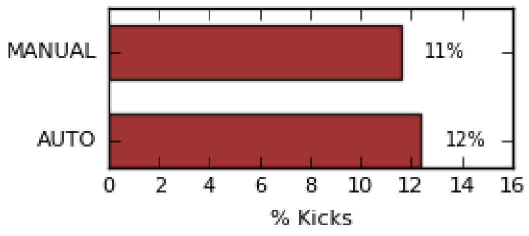


Figure 8: % kicks over Transmissions

Surprisingly in Figure 8 we have not noticed any difference between automatic or manual transmission in terms of defective vehicles. This means that in the last decades, automatic transmissions have made a progress to reduce electrical problems.

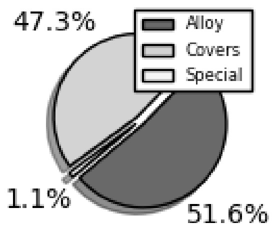


Figure 9: WheelType distribution

From Figure 9, special wheels have a small impact on our samples, but in proportion they have a high rate of kicks (Figure 10). We have supposed that vehicles with that kind of wheel have a risk to face mechanical problems due to the complexity of assembling that type of wheel.

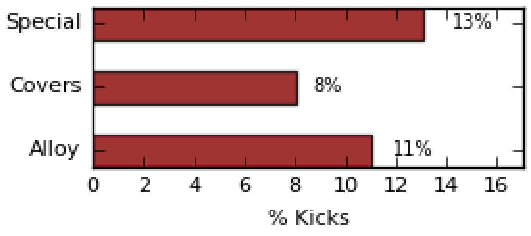


Figure 10: % kicks over WheelType

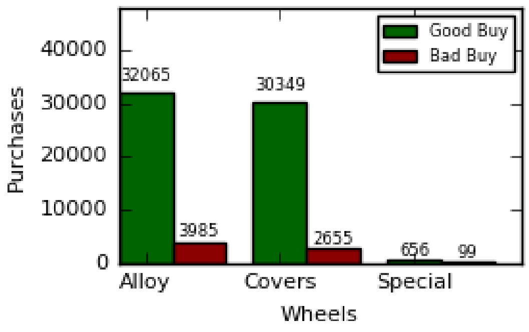


Figure 11: IsBadBuy distribution over WheelType

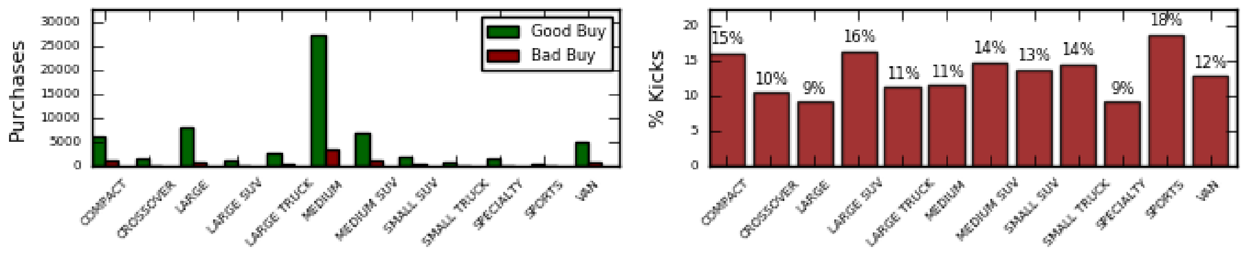


Figure 12: IsBadBuy distribution and % of kicks over Sizes

From Figure 12 most purchases involve medium size vehicles, which is realistic because it includes an high range of users. Furthermore, we have observed a fact about sport cars: they have a 18% rate of being a Bad Buy because they are more subjected to mechanical stress due to their sporty nature. In *VehOdo* examination, to simplify, we have arranged our samples within ranges of 10k values (i.e. 0-10k, 10-20k, ..). We have noticed in Figure 13 a Gaussian distribution of *IsBadBuy* over this attribute and a rate of kicks that has a growing trend in correlation of how many kilometers the vehicle has. As expected, it is normal that a car with a high amount of kilometers can run into mechanical problems.

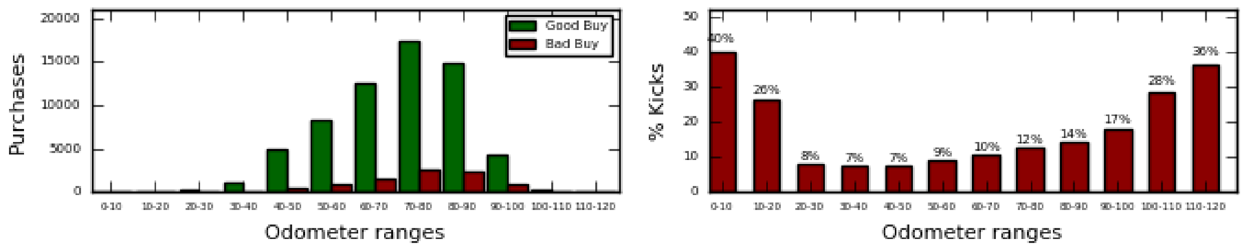


Figure 13: IsBadBuy distribution and % of kicks over VehOdo

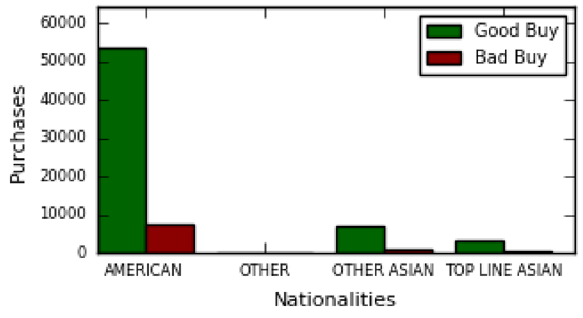


Figure 14: IsBadBuy distribution over Nationality

As expected from an American data set, in Figure 14 most of the purchases have their manufacturer based in USA, so our data set is unbalanced in favour of American Nationality. Linear distribution in Figure 15 shows us that *Nationality* attribute does not affect vehicle conditions, but we have taken carefully this result due to previous unbalanced situation.

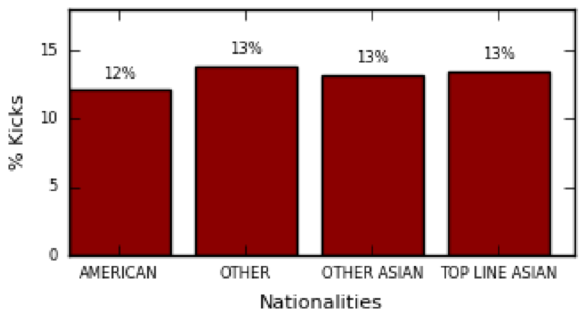


Figure 15: % kicks over Nationality

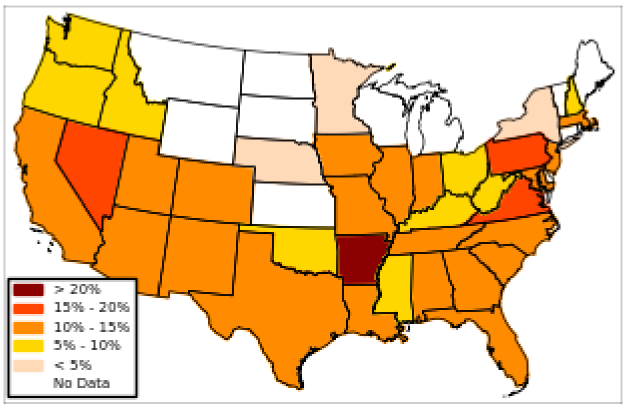


Figure 16: Map % of kicks over different USA States (VNST)

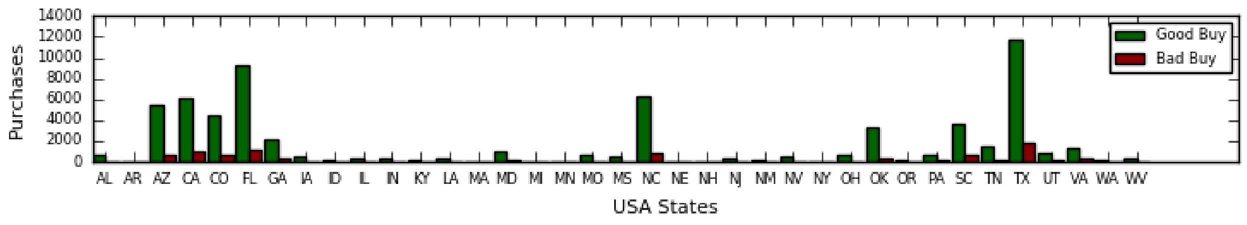


Figure 17: % of kicks over different USA States (VNST)

Figure 16 and Figure 17 show us the diffusion of purchases in the various states. Ranking is led by Texas with 18,63% of total entries, followed by Florida (14,31%), California (9,72%), North Carolina (9,65%) and Arizona (8,46%).

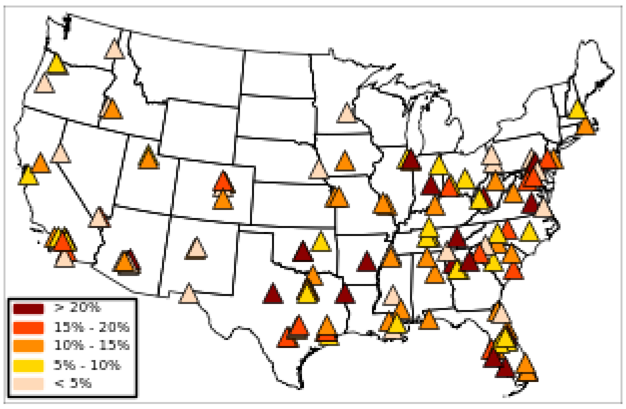


Figure 18: Map % of kicks over different USA Cities (VNZIP1)

Figure 18 shows the spread of purchases along different USA Cities recognized by their postal code. *VNZIP1* is closely related to previous attribute *VNST* and which is way too skewed to be taken into consideration.

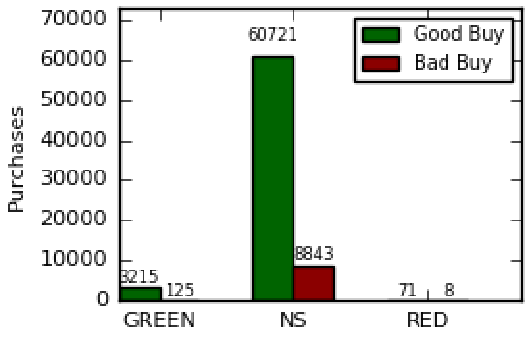


Figure 19: IsBadBuy distribution over AUCGUART

In Figure 19 there is a high amount of missing values: a common hypothesis is to consider NaN values as an additional guarantee level to those which already exist, a yellow guarantee. Figure 20 proves that purchases with green guarantee level have low chances of being a kick, while yellow or red guarantee have a high rate of kicks.

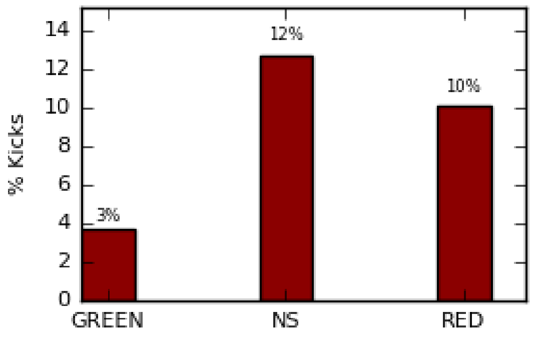


Figure 20: % of kicks over AUCGUART

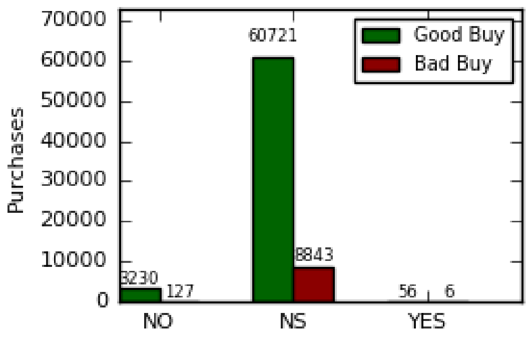


Figure 21: IsBadBuy distribution over PRIMEUNIT

Figure 21 shows another attribute which has mainly missing values, as the previous one. *PRIMEUNIT* is totally inaccurate and does not provide any useful information. Later on, *AUCGUART* and *PRIMEUNIT* will be analyzed more thoroughly.

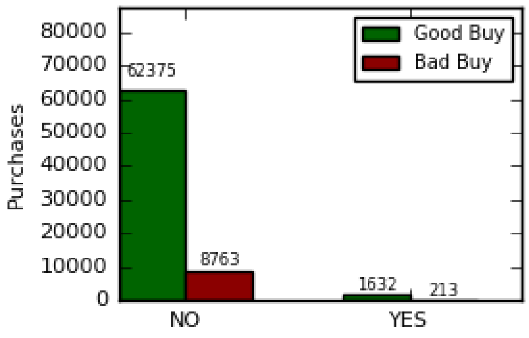


Figure 22: IsBadBuy distribution over IsOnlineSale

From Figure 22 in 97,47% of purchases, the vehicle is not originally acquired online. This attribute is too unbalanced and does not provide us any helpful information for our predictions.

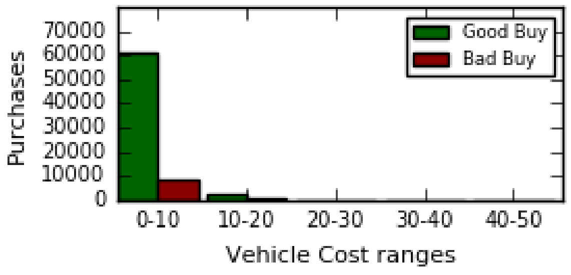


Figure 23: IsBadBuy distribution over VehBCost

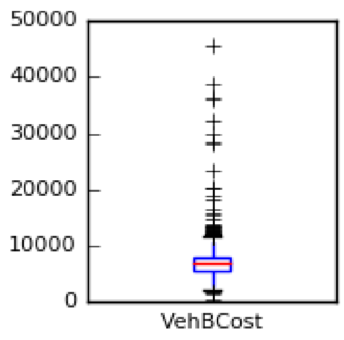


Figure 24: Vehicle Cost BoxPlot

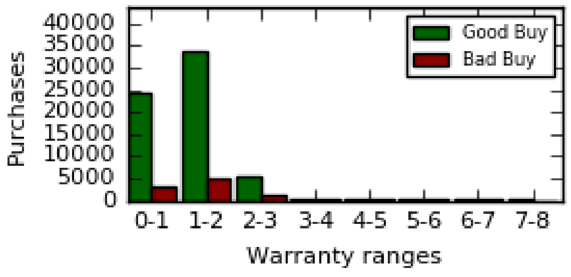


Figure 25: IsBadBuy distribution over WarrantyCost

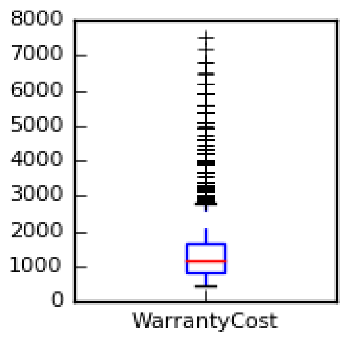


Figure 26: Warranty Cost BoxPlot

VehBCost and *WarrantyCost* attributes in Figure 23 and in Figure 25 show a skewed right distribution, probably for a presence of outliers values. Figure 24 and Figure 26 reinforce this supposition.

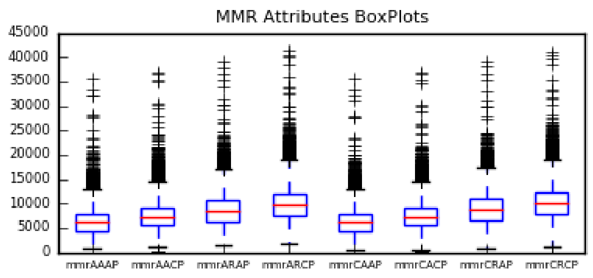
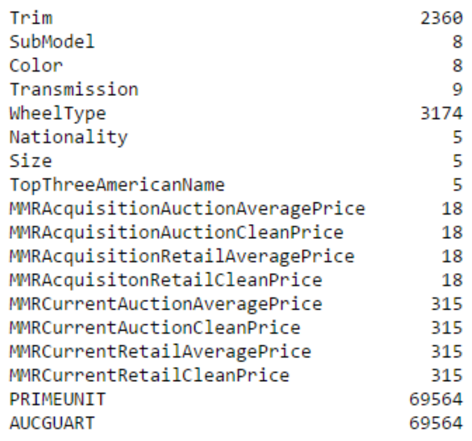


Figure 27: MMR Attributes BoxPlots

MMR attributes are *Manheim Market Report*[1] prices: indicators for similar vehicles and their recommended price. They are subject to change over time and hardly match the exact sale price of a vehicle. Because of this considerations, we have considered an high probability of outliers values.



```
Trim                                2360
SubModel                               8
Color                                  8
Transmission                           9
WheelType                           3174
Nationality                            5
Size                                   5
TopThreeAmericanName                   5
MMRAcquisitionAuctionAveragePrice     18
MMRAcquisitionAuctionCleanPrice       18
MMRAcquisitionRetailAveragePrice      18
MMRAcquisitonRetailCleanPrice         18
MMRCurrentAuctionAveragePrice        315
MMRCurrentAuctionCleanPrice          315
MMRCurrentRetailAveragePrice         315
MMRCurrentRetailCleanPrice           315
PRIMEUNIT                          69564
AUCGUART                           69564
```

Figure 28: Missing values

Assessing data quality. We have inspected the distribution of **missing values** along the data set attributes. As seen before, only two attributes have an high rate of missing values: *PRIMEUNIT* and *AUCGUART* with both 69564 values, so we did not hesitate to exclude them for poor data quality.

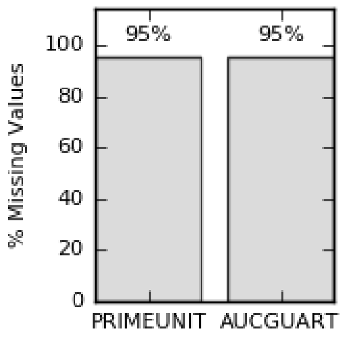
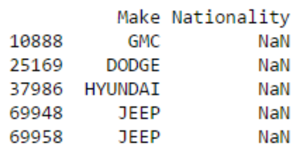


Figure 29: PRIMEUNIT and AUCGUART

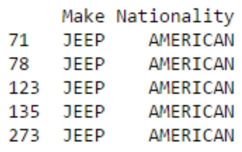
[1]https://www.manheim.com

Attributes with a low quantity of missing values have been replaced with 'NS' value (Not Specified). In addition, we have discovered that missing values of variables like *Nationality* and *Size* can be easily replaced with their correct values. The first one is due to the nationality of the vehicle manufacturer (i.e. JEEP is an American Manufacturer) and the second one by the correlation with the vehicle model (i.e. The GMC Sierra is a Large Truck). These replacements can be done with the associations rules of the third chapter.



```
          Make Nationality
10888      GMC         NaN
25169    DODGE         NaN
37986  HYUNDAI         NaN
69948     JEEP         NaN
69958     JEEP         NaN
```

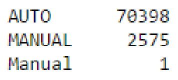
Figure 30: Missing values of Nationality



```
     Make Nationality
71   JEEP    AMERICAN
78   JEEP    AMERICAN
123  JEEP    AMERICAN
135  JEEP    AMERICAN
273  JEEP    AMERICAN
```

Figure 31: Samples of JEEP Nationalities

Variables transformations. An **ambiguity** has been corrected in *Transmission* attribute: a value with 'Manual' label has been replaced with 'MANUAL' simply via an uppercase transformation of nominal data.



```
AUTO      70398
MANUAL     2575
Manual        1
```

Figure 32: Ambiguity in Transmission

Afterwards, we have removed both useless attributes and the ones which have a negative impact for our prediction:

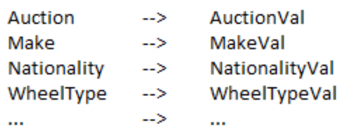
- Identifiers like *RefId* or *BYRNO*
- Dates as *PurchDate*
- *VNZIP1* in favor of *VNST* that is more generic and less confused
- Vehicle-specific information as *Model*, *Trim*, *SubModel* and *Color*
- *Transmission* is unbalanced in favor of automatic (Figure 6).
- *Size* is unbalanced in favor of medium size (Figure 12)
- *TopThreeAmericanName* is useless
- All the MMR indicators, because, as seen previously, they do not match exactly the real sale price, but are only virtual indexes.
- *IsOnlineSale* is unbalanced in favor of 'not originally purchased online' (Figure 22).

Elimination of redundant variables. We have identified that *VehYear* and *VehicleAge* have an high correlation, so we decided to filter out *VehYear* in favor of *VehicleAge*; Same situation with *WheelTypeID* and *WheelType*, where we have kept the last one.

Feature Selection. Before going on clustering analysis, we made a feature selection about attributes seen previously. In this process we used the recursive feature elimination (RFE) method, that works by recursively removing attributes and building a model on those attributes that remain. It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target variable to be predicted, which in our case is *IsBadBuy*. The most important features sorted in descending order by their ranking are: *VehicleAge*, *VehOdo*, *WarrantyCost* and *VehBCost*. From now on, we will consider these attributes as our best predictors.

Clustering

In clustering analysis, non-numeric data need to be converted to numeric, in order to allow clustering algorithms to process information correctly.



```
Auction       -->   AuctionVal
Make          -->   MakeVal
Nationality   -->   NationalityVal
WheelType     -->   WheelTypeVal
...           -->   ...
```

Figure 33: non-numeric data mapping examples

Afterwards, during the data pre-processing stage, we have normalized data with Feature Scaling method (also know as Min-Max Scaler). Data have been scaled in a range between 0 and 1, in order to standardize our attributes. In this section we have examined through clustering analysis our best predictors, according to considerations made previously in Feature Selection section. We expect to find some similarities or correlations between values of these attributes.

Clustering Analysis by K-means. The main problem that we have encountered was to decide an appropriate number of clusters to be used as a parameter for K-means algorithm. For this purpose we have taken in consideration the **Elbow method** and the **Silhouette Score**.

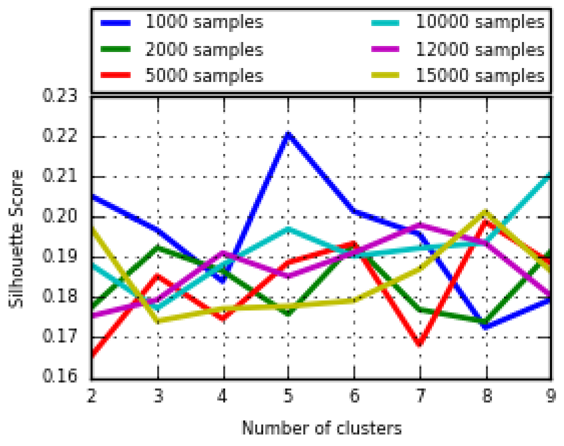


Figure 34: Selecting clusters with Silhouette Score

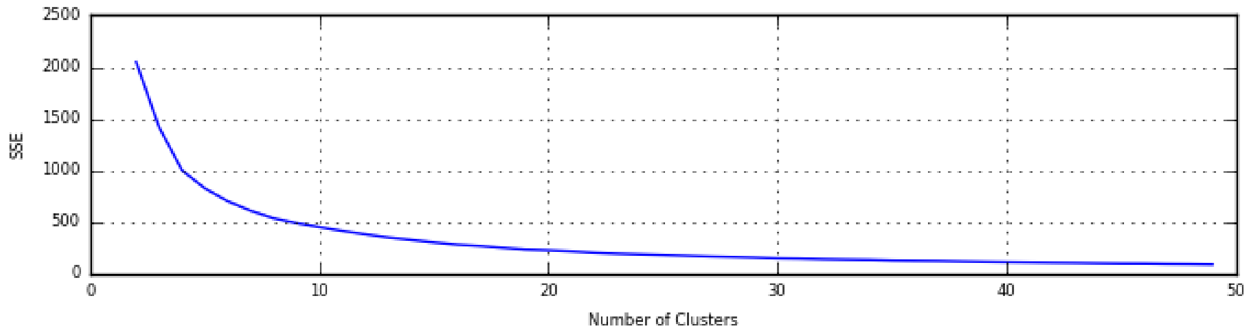


Figure 35: Selecting clusters with Elbow method

The curve obtained from Elbow method in Figure 35 forms a good angle in x=8, that suggests us to choose 8 clusters as a parameter for the algorithm. Not so much different results with silhouette score in Figure 34, plots with 12k and 15k samples reach the best score near 8 clusters. At this point, with a number of cluster equal to 8, we have observed the behavior of K-means algorithm. Figure 36 shows an analysis of cluster's centroids across a selection of attributes.

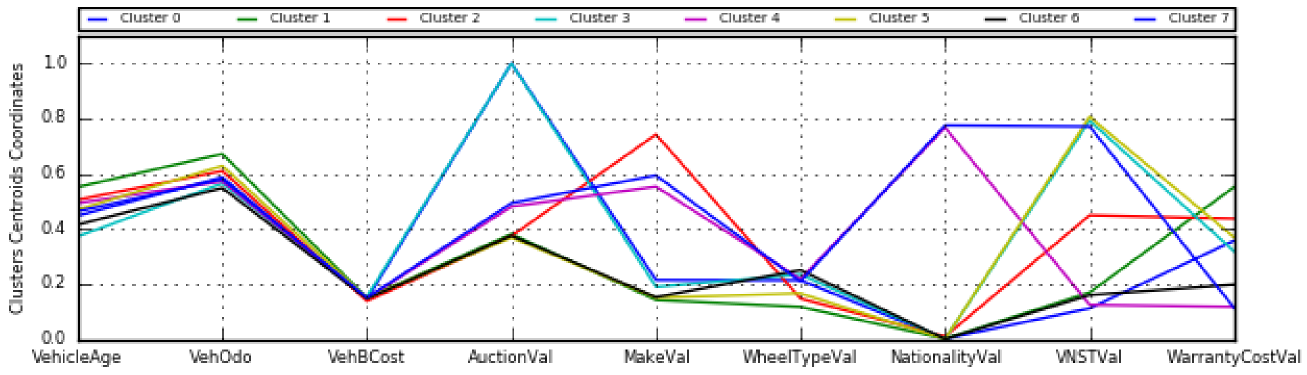


Figure 36: Comparing each cluster centroid over attributes

In the first stage, we have explored *VehOdo* and *WarrantyCost* variables. For this two, the outcome can be visualized in Figure 37. In this case, the algorithm has well clustered all values in eight groups. In support of this consideration comes the silhouette score: 0.41.

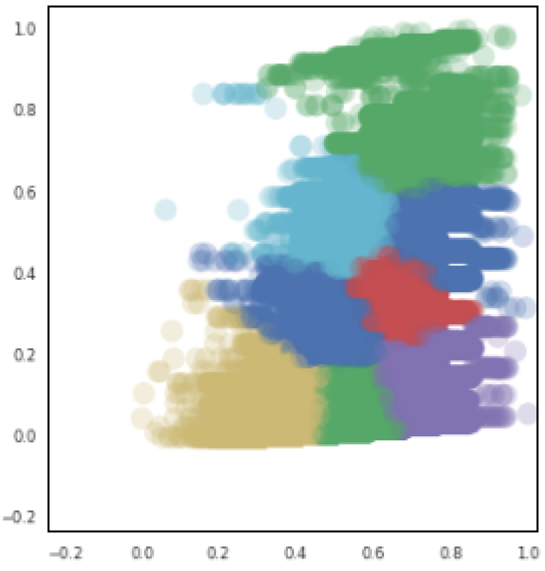


Figure 37: Clusters found by K-means

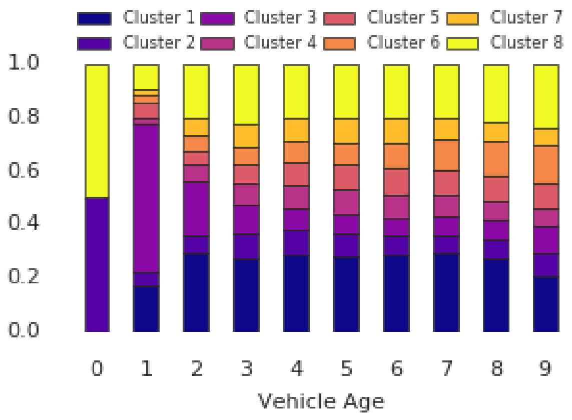


Figure 38: Data clustered over VehicleAge

Afterwards, we have considered another important attribute: *VehicleAge*. In Figure 38 we have noticed a good values distribution across all clusters, especially between two and nine years of age. Another well-distributed variable is *WheelType*, as we can see in Figure 39. In opposition, variables as *Auction* and *Nationality* have a bad distribution of values within clusters (Figure 40 and Figure 41).

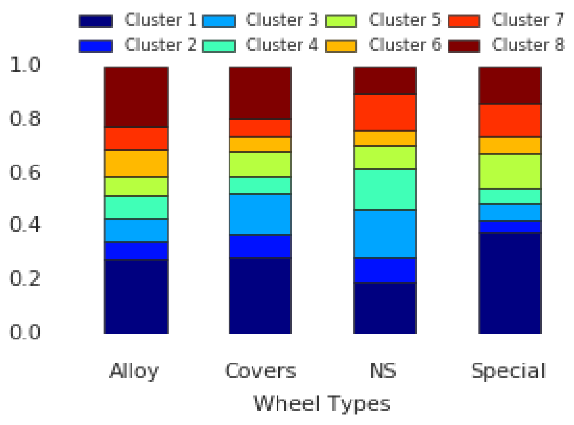


Figure 39: Data clustered over WheelType

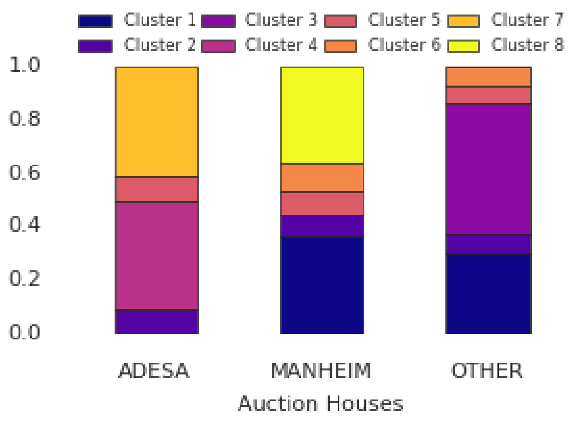


Figure 40: Data clustered over Auction

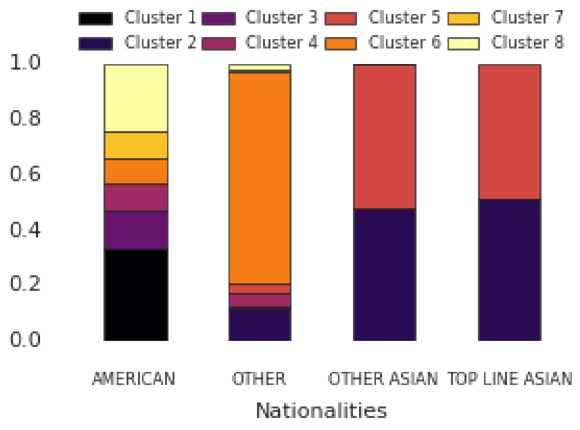


Figure 41: Data clustered over Nationality

At the end, we have explored our three best predictors, *VehicleOdo*, *VehBCost* and *WarrantyCost* together. Again, the algorithm has done a good job in clustering values of these attributes. However, we have noticed a small decline in silhouette coefficient with a result of 0.36, due to some remaining outliers, as Figure 42 shows.

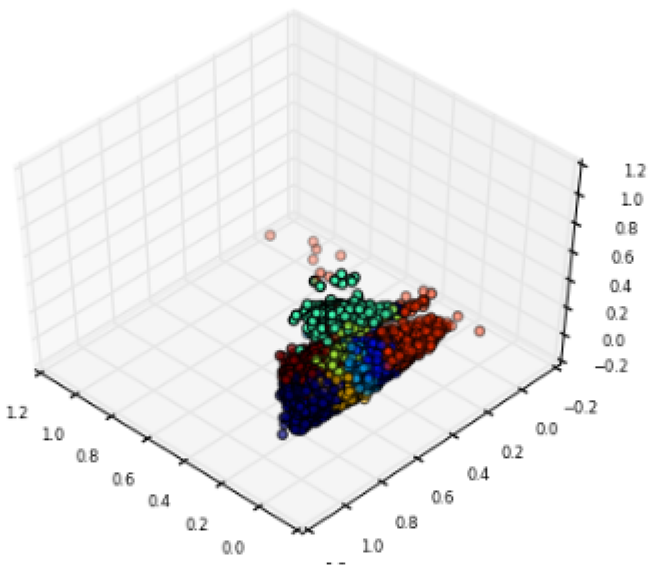


Figure 42: Clusters found by K-Means

Analysis by hierarchical clustering. All our experiments have been conducted with an agglomerative strategy and with the same Euclidean metric to compute pairwise distances, but changing the linkage criteria. The linkage criterion that we used are displayed in the following graphs: a single-linkage criterion in Figure 43, an average-linkage criterion in Figure 44, a complete-linkage criterion in Figure 45, a weighted-linkage criterion in Figure 46 and a ward-linkage criterion in Figure 47. The only rule that we have imposed to the algorithm is to find eight clusters. Furthermore, we made some silhouette scoring with 15k samples for memory limitations that will be discussed later, with the following results:

- Average-linkage criterion: 0.145963230755
- Ward-linkage criterion: 0.201987857049
- Complete-linkage criterion: 0.0220693301629

As we have expected, silhouette scoring is not so high. Fortunately, we did not go below zero, but with these values a good clustering quality cannot be reached.

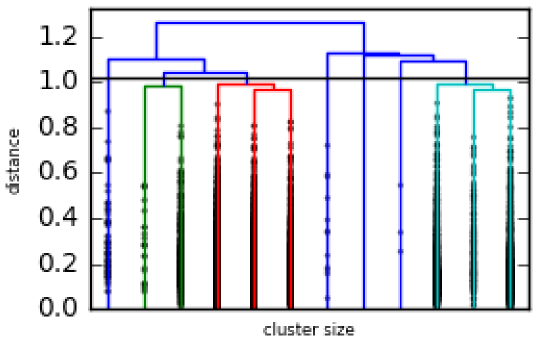


Figure 43: Dendogram result from Euclidean metric and Single Linkage Criteria

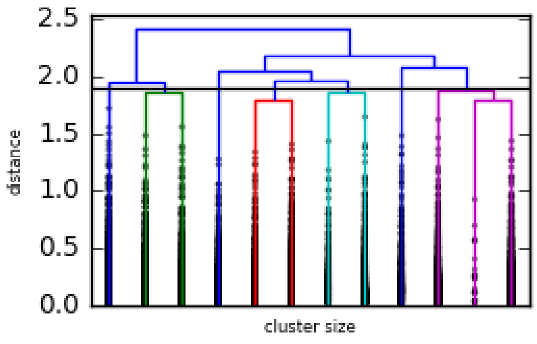


Figure 44: Dendogram result from Euclidean metric and Complete Linkage criterion

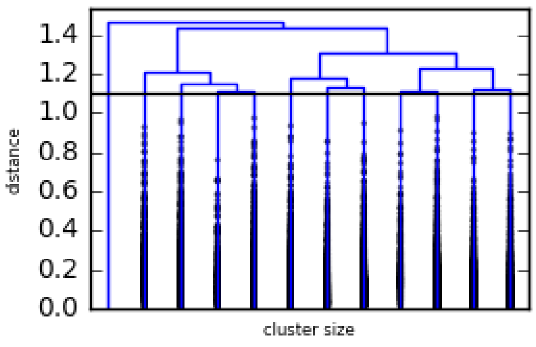


Figure 46: Dendogram result from Euclidean metric and Weighted Linkage criterion

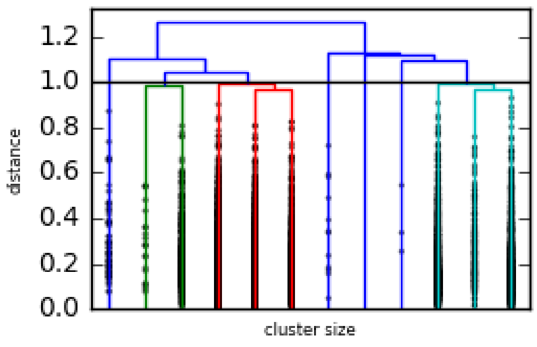


Figure 45: Dendogram result from Euclidean metric and Average Linkage criterion

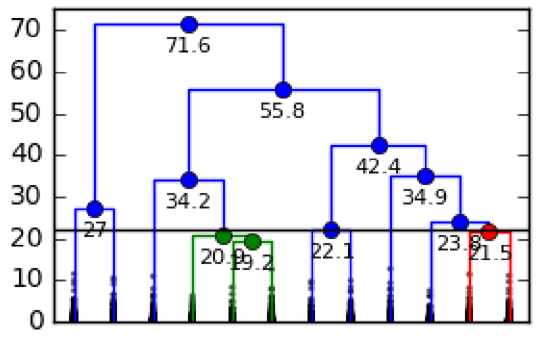


Figure 47: Dendogram result from Euclidean metric and Ward Linkage criterion

Analysis by density-based clustering. During our observations, we have noticed a good density-based clustering with two attributes: *VehBCost* and *WarrantyCost*, considered the best predictors for our model along with *VehicleAge* and *VehOdo*. Noteworthy the **silhouette score** obtained with the first two: 0.488465325813, which means that the objects match well their own cluster. However, one of the best advantages of DBSCAN is to find outliers, as we can see in Figure 48 (black points). Differently from K-means clustering analysis, with *VehOdo* in addition to the previous two, we got a silhouette score below zero, that means a bad clustering by the algorithm, due to the probable presence of outliers and memory problems due to small size of samples, as seen previously in hierarchical clustering.

Conclusions on clustering analysis During our evaluation process, we have encountered some problems due to the dimension of the data set provided by kaggle. In fact, using the entire data set, we have always gone out of memory or had some issues with hierarchical and density-based clustering. This kind of analysis works well with small data set, so we were forced to partition our data in subsets of about 20k - 30k elements, aware of introducing some error in our evaluation. The complexity of agglomerative clustering and density-based clustering (as DBSCAN) is caused by the computation of a distance matrix (pairwise distances between data set objects), that always takes $O(N^2)$. Furthermore, we had some problems due to the method used for evaluating this kind of clustering analysis: the silhouette score. The silhouette method requires to calculate a distance matrix and uses a huge amount of memory, beyond our hardware capabilities. This obligate us to set a sample size of 15k or max 20k while calculating the score. The results we have achieved from this kind of analysis are not considered optimal and need to be taken carefully. In K-means clustering analysis, we have found the best results because this algorithm works well in large data sets. Initially, we have applied K-means to all data set attributes and we found several confirmations about our previous hypothesis during data understanding process. Later, we have excluded some variables and re-applied it multiple times, trying to find similarities and correlations in order to have a good overview about our data.

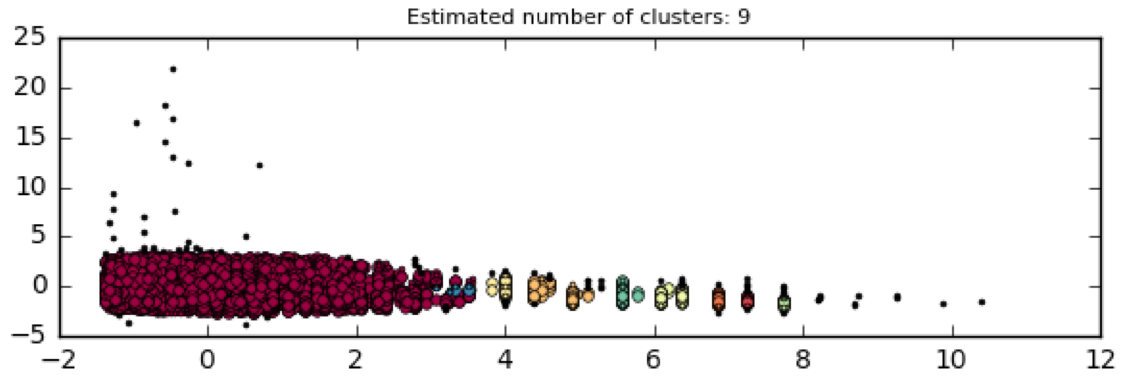


Figure 48: DBSCAN with VehBCost and WarrantyCost

Association Rules Mining

In this section we are going to present the frequent patterns and association rules we have extracted from the training data set. We have obtained different results changing some parameters, in order to obtain useful informations.

Frequent patterns extraction. We have tried to extract some frequent patterns using a minimum support equal to 90%. Below are listed the frequent patterns we have obtained as result, for each pattern we have in parenthesis the support expressed as a percentage:

- NS_AUCGUART NS_PRIMEUNIT (95.3153)
- NS_AUCGUART AUTO_Transmission NS_PRIMEUNIT (92.0584)
- NS_AUCGUART AUTO_Transmission (92.0584)
- NS_AUCGUART 0_IsOnlineSale NS_PRIMEUNIT (92.9792)
- NS_AUCGUART 0_IsOnlineSale (92.9792)
- NS_PRIMEUNIT AUTO_Transmission (92.0584)
- NS_PRIMEUNIT 0_IsOnlineSale (92.9792)
- AUTO_Transmission 0_IsOnlineSale (94.0562)

As we can see we got only few and useless patterns because, as we have showed in the data understanding part, the variables *AUCGUARD* and *PRIMEUNIT* have 95% of records that are not specified. For this reason it is obvious that these two variables are in patterns with minimum support equal to 90%, with *NS* (Not Specified) value. Instead, the variable *IsOnlineSale* has a majority of *'No'* in the data set and the variable *Transmission* have 96.5% of *AUTOMATIC* values. We can conclude that these rules are not helpful to get any information on the data set.

From this result we have understood that if we what to get some useful frequent patterns, we have to remove from this analysis the following variables: *AUCGUARD*, *PRIMEUNIT*, *Transmission* and *IsOnlineSale*.

After this modifications, we have extracted the maximal patterns with the remaining variables using a minimum support equal to 10 and minimum number of items equal to 3. This because with minimum number of items equal to 2 we have obtained obvious patterns, already explained in the data understanding part. With these parameters we have obtained the following patterns:

- CHRYSLER_Make AMERICAN_Nationality 0_IsBadBuy (10.56)
- LARGE_Size AMERICAN_Nationality 0_IsBadBuy (10.734)
- SE_Trim AMERICAN_Nationality 0_IsBadBuy (10.6573)
- LS_Trim CHEVROLET_Make AMERICAN_Nationality 0_IsBadBuy (12.1261)
- BLUE_Color AMERICAN_Nationality 0_IsBadBuy (10.5024)
- FL_VNST AMERICAN_Nationality 0_IsBadBuy (10.0845)
- FORD_Make Alloy_WheelType AMERICAN_Nationality (11.0409)
- FORD_Make AMERICAN_Nationality 0_IsBadBuy (13.1031)
- WHITE_Color AMERICAN_Nationality 0_IsBadBuy (12.5536)
- DODGE_Make AMERICAN_Nationality 0_IsBadBuy (15.864)
- 5_VehicleAge AMERICAN_Nationality 0_IsBadBuy (12.4166)
- TX_VNST MANHEIM_Auction AMERICAN_Nationality (10.0352)
- TX_VNST MANHEIM_Auction 0_IsBadBuy (10.5847)
- TX_VNST AMERICAN_Nationality 0_IsBadBuy (13.4113)
- Bas_Trim 4D-SEDAN_SubModel AMERICAN_Nationality (10.0667)
- Bas_Trim 4D-SEDAN_SubModel 0_IsBadBuy (10.4106)
- Bas_Trim Covers_WheelType 0_IsBadBuy (10.3599)
- Bas_Trim AMERICAN_Nationality 0_IsBadBuy (14.08)
- 60000_Odometer MANHEIM_Auction 0_IsBadBuy (10.2243)
- 60000_Odometer AMERICAN_Nationality 0_IsBadBuy (14.2718)
- ADESA_Auction AMERICAN_Nationality 0_IsBadBuy (13.6511)
- SILVER_Color AMERICAN_Nationality 0_IsBadBuy (14.8569)
- 4D-SEDAN_SubModel MEDIUM_Size 0_IsBadBuy (12.4769)
- 4D-SEDAN_SubModel Covers_WheelType 0_IsBadBuy (12.2179)
- 4D-SEDAN_SubModel AMERICAN_Nationality 0_IsBadBuy (13.7251)
- 1500_WarrantyCost Alloy_WheelType AMERICAN_Nationality 0_IsBadBuy (10.4243)
- 1500_WarrantyCost MANHEIM_Auction AMERICAN_Nationality 0_IsBadBuy (10.7655)
- 3_VehicleAge Covers_WheelType 0_IsBadBuy (11.4616)
- 3_VehicleAge MANHEIM_Auction AMERICAN_Nationality 0_IsBadBuy (10.0229)
- 4_VehicleAge Alloy_WheelType 0_IsBadBuy (10.586)

- 4_VehicleAge MANHEIM_Auction AMERICAN_Nationality 0_IsBadBuy (10.1024)
- CHEVROLET_Make Covers_WheelType AMERICAN_Nationality 0_IsBadBuy (11.5739)
- CHEVROLET_Make Alloy_WheelType AMERICAN_Nationality (10.0599)
- CHEVROLET_Make MANHEIM_Auction AMERICAN_Nationality 0_IsBadBuy (12.2837)
- 80000_Odometer Alloy_WheelType AMERICAN_Nationality 0_IsBadBuy (10.7765)
- 80000_Odometer MANHEIM_Auction AMERICAN_Nationality 0_IsBadBuy (10.5573)
- OTHER_Auction MEDIUM_Size AMERICAN_Nationality (10.0571)
- OTHER_Auction MEDIUM_Size 0_IsBadBuy (10.8231)
- OTHER_Auction Covers_WheelType AMERICAN_Nationality 0_IsBadBuy (10.2709)
- 70000_Odometer Covers_WheelType 0_IsBadBuy (10.1517)
- 70000_Odometer Alloy_WheelType AMERICAN_Nationality 0_IsBadBuy (11.1697)
- 70000_Odometer MANHEIM_Auction AMERICAN_Nationality 0_IsBadBuy (12.1933)
- 1000_WarrantyCost MEDIUM_Size AMERICAN_Nationality 0_IsBadBuy (12.8866)
- 1000_WarrantyCost Covers_WheelType AMERICAN_Nationality 0_IsBadBuy (10.9368)
- 1000_WarrantyCost Alloy_WheelType MANHEIM_Auction (10.3736)
- 1000_WarrantyCost Alloy_WheelType AMERICAN_Nationality 0_IsBadBuy (13.4744)
- 1000_WarrantyCost MANHEIM_Auction AMERICAN_Nationality 0_IsBadBuy (14.8651)
- 500_WarrantyCost MEDIUM_Size Covers_WheelType 0_IsBadBuy (13.1866)
- 500_WarrantyCost MEDIUM_Size MANHEIM_Auction (10.671)
- 500_WarrantyCost MEDIUM_Size AMERICAN_Nationality 0_IsBadBuy (13.1839)
- 500_WarrantyCost Covers_WheelType AMERICAN_Nationality 0_IsBadBuy (12.1795)
- 500_WarrantyCost Alloy_WheelType AMERICAN_Nationality (10.1366)
- 500_WarrantyCost Alloy_WheelType 0_IsBadBuy (13.0729)
- 500_WarrantyCost MANHEIM_Auction AMERICAN_Nationality 0_IsBadBuy (10.9025)
- MEDIUM_Size Covers_WheelType MANHEIM_Auction 0_IsBadBuy (11.2163)
- MEDIUM_Size Covers_WheelType AMERICAN_Nationality 0_IsBadBuy (16.8985)
- MEDIUM_Size Alloy_WheelType AMERICAN_Nationality 0_IsBadBuy (11.3026)
- MEDIUM_Size MANHEIM_Auction AMERICAN_Nationality 0_IsBadBuy (15.1378)
- Covers_WheelType MANHEIM_Auction AMERICAN_Nationality 0_IsBadBuy (18.3234)
- Alloy_WheelType MANHEIM_Auction AMERICAN_Nationality 0_IsBadBuy (22.1284)

Some of this patterns are not so useful, such as this one:

> DODGE_Make
> AMERICAN_Nationality
> 0_IsBadBuy

This pattern is not so useful because all the dodge car are American. It's normal that a lot of them are good buy because, as we have seen in the data understanding part, a lot of cars in the data set are good buy. In the association rule section this kind of rules will probably be useful considering also the confidence; In the next pages the association rules will be explained in details.

Luckily, there are not only useless pattern but also useful ones, like the following one:

> FORD_Make Alloy_WheelType
> AMERICAN_Nationality

This pattern allow us to say that in the USA, the Ford company does not sell mainly models like Fiesta or Focus as in Europe; Indeed these cars, normally, have cover wheel type, but probably in USA are sold models like Mustang or bigger ones with alloy wheel. This is reasonable because in the USA, Americans normally prefer bigger cars in comparison to Europeans, due to the fact that they have bigger roads than the ones that are in Europe.

Another interesting pattern is the following one:

> TX_VNST MANHEIM_Auction
> AMERICAN_Nationality

This is an interesting pattern because a lot of cars sold by Manheim Auction that came from Texas are American cars. An explanation of this, could be that people living in Texas tends to be more conservative; Preferring to buy made in America cars rather than foreigners brands. But this statement is invalidated by the following pattern:

> FL_VNST
> AMERICAN_Nationality
> 0_IsBadBuy

Indeed, even if Florida state is a less conservative state compared to Texas, we have extracted the pattern above. This is explained by the high number of American cars in the data set. Before making other statements, we have to generate the association rules of the following patterns, and evaluate the confidence parameter.

> Bas_Trim 4D-SEDAN_SubModel
> AMERICAN_Nationality

The above pattern tells us that a lot of American cars in the data set have a base trim, this means that there are no additional items added to the standard car. The sub model is *4D Sedan*, where the *4D* probably stands for 4 Doors, in USA they probably quantify the number of doors without counting also the boot, as we do in Europe.

CHEVROLET_Make
MANHEIM_Auction
AMERICAN_Nationality
0_IsBadBuy

The above pattern is interesting because it shows that a lot of Chevrolet cars are bought by the Manheim auction. In fact there are no other patterns telling us that Manheim auction bought different makes. So for this reason, from the data set, it can be said that the preferred make of the Manheim auction is the Chevrolet.

OTHER_Auction MEDIUM_Size
AMERICAN_Nationality

The last pattern is interesting because it tells us that for auctions, different from Manheim and Adusa, the cars have normally a medium size and are American.

We have tried with different parameters but we did not get any significant result. From this analysis we have not reached any useful pattern, also because the patterns have any confidence parameter useful to deduce information.

Association rules extraction. In this section we are going to present the association rules extracted from the training data set. Afterwards, we are going to explain the most interesting ones. For the same reasons that we have explained in the frequent pattern section, we have removed the following variables: *AUCGUARD*, *PRIMEUNIT*, *Transmission* and *IsOnlineSale*.

At first, we have tried to change the parameters but at the end we have decided to use as minimum support 10 and minimum confidence 90. We obtained the following association rules, after each one of them, we have reported the lift and confidence measures:

- [2_VehicleAge, AMERICAN_Nationality] → 0_IsBadBuy lift 106.881 conf 93.7357
- [2_VehicleAge] → 0_IsBadBuy lift 106.764 conf 93.6336
- [CHRYSLER_Make, 0_IsBadBuy] → AMERICAN_Nationality lift 119.589 conf 100.0
- [CHRYSLER_Make] → AMERICAN_Nationality lift 119.589 conf 100.0
- [LARGE_Size, AMERICAN_Nationality] → 0_IsBadBuy lift 104.158 conf 91.3479
- [LARGE_Size, 0_IsBadBuy] → AMERICAN_Nationality lift 116.641 conf 97.5349
- [LARGE_Size] → AMERICAN_Nationality lift 115.887 conf 96.904
- [LARGE_Size] → 0_IsBadBuy lift 103.484 conf 90.7571
- [50000_Odometer, AMERICAN_Nationality] → 0_IsBadBuy lift 103.808 conf 91.041
- [50000_Odometer] → 0_IsBadBuy lift 103.412 conf 90.6936
- [SE_Trim, 0_IsBadBuy] → AMERICAN_Nationality lift 113.449 conf 94.8652
- [SE_Trim] → AMERICAN_Nationality lift 112.63 conf 94.1806
- [LS_Trim, CHEVROLET_Make, AMERICAN_Nationality] → 0_IsBadBuy lift 104.473 conf 91.6244
- [LS_Trim, CHEVROLET_Make, 0_IsBadBuy] → AMERICAN_Nationality lift 119.589 conf 100.0
- [LS_Trim, AMERICAN_Nationality, 0_IsBadBuy] → CHEVROLET_Make lift 413.742 conf 97.7793
- [LS_Trim, CHEVROLET_Make] → AMERICAN_Nationality lift 119.589 conf 100.0
- [LS_Trim, AMERICAN_Nationality] → CHEVROLET_Make lift 412.089 conf 97.3886
- [LS_Trim, CHEVROLET_Make] → 0_IsBadBuy lift 104.473 conf 91.6244
- [LS_Trim, 0_IsBadBuy] → CHEVROLET_Make lift 403.532 conf 95.3664
- [LS_Trim] → CHEVROLET_Make lift 401.72 conf 94.9381
- [LS_Trim, AMERICAN_Nationality] → 0_IsBadBuy lift 104.056 conf 91.2583
- [LS_Trim, 0_IsBadBuy] → AMERICAN_Nationality lift 116.638 conf 97.5323
- [LS_Trim] → AMERICAN_Nationality lift 116.58 conf 97.4838
- [LS_Trim] → 0_IsBadBuy lift 104.004 conf 91.2129
- [FORD_Make, Alloy_WheelType] → AMERICAN_Nationality lift 119.589 conf 100.0
- [FORD_Make, 0_IsBadBuy] → AMERICAN_Nationality lift 119.589 conf 100.0
- [FORD_Make] → AMERICAN_Nationality lift 119.589 conf 100.0
- [DODGE_Make, 0_IsBadBuy] → AMERICAN_Nationality lift 119.579 conf 99.9914
- [DODGE_Make] → AMERICAN_Nationality lift 119.58 conf 99.9923
- [Bas_Trim, Covers_WheelType] → 0_IsBadBuy lift 103.138 conf 90.4534
- [ADESA_Auction, Alloy_WheelType] → 0_IsBadBuy lift 104.027 conf 91.2333
- [4D-SEDAN_SubModel, Covers_WheelType] → 0_IsBadBuy lift 103.739 conf 90.9805
- [1500_WarrantyCost, Alloy_WheelType, 0_IsBadBuy] → AMERICAN_Nationality lift 113.403 conf 94.8274
- [1500_WarrantyCost, Alloy_WheelType] → AMERICAN_Nationality lift 112.841 conf 94.357
- [1500_WarrantyCost, MANHEIM_Auction, 0_IsBadBuy] → AMERICAN_Nationality lift 115.304 conf 96.4167
- [1500_WarrantyCost, MANHEIM_Auction] → AMERICAN_Nationality lift 114.858 conf 96.0434
- [1500_WarrantyCost, 0_IsBadBuy] → AMERICAN_Nationality lift 115.54 conf 96.6144
- [1500_WarrantyCost] → AMERICAN_Nationality lift 115.043 conf 96.1983
- [3_VehicleAge, Covers_WheelType] → 0_IsBadBuy lift 107.254 conf 94.0627
- [3_VehicleAge, MANHEIM_Auction, AMERICAN_Nationality] → 0_IsBadBuy lift 106.252 conf 93.1847
- [3_VehicleAge, MANHEIM_Auction] → 0_IsBadBuy lift 105.895 conf 92.8716
- [3_VehicleAge, AMERICAN_Nationality] → 0_IsBadBuy lift 104.974 conf 92.0639
- [3_VehicleAge] → 0_IsBadBuy lift 104.695 conf 91.8186
- [4_VehicleAge, Covers_WheelType] → 0_IsBadBuy lift 104.769 conf 91.8838
- [4_VehicleAge, Alloy_WheelType] → 0_IsBadBuy lift 104.057 conf 91.2592
- [4_VehicleAge, MANHEIM_Auction, AMERICAN_Nationality] → 0_IsBadBuy lift 102.862 conf 90.2117

- [4_VehicleAge, MANHEIM_Auction] → 0_IsBadBuy lift 102.849 conf 90.2
- [CHEVROLET_Make, Covers_WheelType, AMERICAN_Nationality] → 0_IsBadBuy lift 106.18 conf 93.1209
- [CHEVROLET_Make, Covers_WheelType, 0_IsBadBuy] → AMERICAN_Nationality lift 119.589 conf 100.0
- [CHEVROLET_Make, Covers_WheelType] → AMERICAN_Nationality lift 119.589 conf 100.0
- [CHEVROLET_Make, Covers_WheelType] → 0_IsBadBuy lift 106.18 conf 93.1209
- [CHEVROLET_Make, Alloy_WheelType, AMERICAN_Nationality] → 0_IsBadBuy lift 104.597 conf 91.7325
- [CHEVROLET_Make, Alloy_WheelType] → AMERICAN_Nationality lift 119.589 conf 100.0
- [CHEVROLET_Make, Alloy_WheelType] → 0_IsBadBuy lift 104.597 conf 91.7325
- [CHEVROLET_Make, MANHEIM_Auction, AMERICAN_Nationality] → 0_IsBadBuy lift 104.106 conf 91.3026
- [CHEVROLET_Make, MANHEIM_Auction, 0_IsBadBuy] → AMERICAN_Nationality lift 119.589 conf 100.0
- [CHEVROLET_Make, MANHEIM_Auction] → AMERICAN_Nationality lift 119.589 conf 100.0
- [CHEVROLET_Make, MANHEIM_Auction] → 0_IsBadBuy lift 104.106 conf 91.3026
- [CHEVROLET_Make, AMERICAN_Nationality] → 0_IsBadBuy lift 102.911 conf 90.2539
- [CHEVROLET_Make, 0_IsBadBuy] → AMERICAN_Nationality lift 119.589 conf 100.0
- [CHEVROLET_Make] → AMERICAN_Nationality lift 119.589 conf 100.0
- [CHEVROLET_Make] → 0_IsBadBuy lift 102.911 conf 90.2539
- [OTHER_Auction, 500_WarrantyCost] → 0_IsBadBuy lift 103.009 conf 90.3397
- [OTHER_Auction, Covers_WheelType, AMERICAN_Nationality] → 0_IsBadBuy lift 108.11 conf 94.8141
- [OTHER_Auction, Covers_WheelType] → 0_IsBadBuy lift 107.818 conf 94.5575
- [70000_Odometer, Covers_WheelType] → 0_IsBadBuy lift 103.618 conf 90.8745
- [1000_WarrantyCost, MEDIUM_Size, 0_IsBadBuy] → AMERICAN_Nationality lift 113.656 conf 95.0384
- [1000_WarrantyCost, MEDIUM_Size] → AMERICAN_Nationality lift 113.434 conf 94.8526
- [1000_WarrantyCost, Covers_WheelType, AMERICAN_Nationality] → 0_IsBadBuy lift 104.613 conf 91.7471
- [1000_WarrantyCost, Covers_WheelType, 0_IsBadBuy] → AMERICAN_Nationality lift 114.525 conf 95.7648
- [1000_WarrantyCost, Covers_WheelType] → AMERICAN_Nationality lift 114.27 conf 95.5519
- [1000_WarrantyCost, Covers_WheelType] → 0_IsBadBuy lift 104.381 conf 91.5431
- [1000_WarrantyCost, MANHEIM_Auction, 0_IsBadBuy] → AMERICAN_Nationality lift 109.248 conf 91.3523
- [1000_WarrantyCost, MANHEIM_Auction] → AMERICAN_Nationality lift 108.641 conf 90.8454
- [1000_WarrantyCost, 0_IsBadBuy] → AMERICAN_Nationality lift 110.034 conf 92.01
- [1000_WarrantyCost] → AMERICAN_Nationality lift 109.416 conf 91.4928
- [500_WarrantyCost, MEDIUM_Size, Covers_WheelType] → 0_IsBadBuy lift 106.934 conf 93.7829
- [500_WarrantyCost, MEDIUM_Size, MANHEIM_Auction] → 0_IsBadBuy lift 103.936 conf 91.1531
- [500_WarrantyCost, MEDIUM_Size, AMERICAN_Nationality] → 0_IsBadBuy lift 104.539 conf 91.6818
- [500_WarrantyCost, MEDIUM_Size] → 0_IsBadBuy lift 103.515 conf 90.7836
- [500_WarrantyCost, Covers_WheelType, AMERICAN_Nationality] → 0_IsBadBuy lift 106.533 conf 93.4307
- [500_WarrantyCost, Covers_WheelType] → 0_IsBadBuy lift 105.623 conf 92.6323
- [500_WarrantyCost, Alloy_WheelType, AMERICAN_Nationality] → 0_IsBadBuy lift 103.312 conf 90.6056
- [500_WarrantyCost, Alloy_WheelType] → 0_IsBadBuy lift 103.412 conf 90.6939
- [MEDIUM_Size, Covers_WheelType, MANHEIM_Auction] → 0_IsBadBuy lift 103.849 conf 91.077
- [MEDIUM_Size, Covers_WheelType, AMERICAN_Nationality] → 0_IsBadBuy lift 105.582 conf 92.597
- [MEDIUM_Size, Covers_WheelType] → 0_IsBadBuy lift 105.408 conf 92.4445
- [Covers_WheelType, MANHEIM_Auction, AMERICAN_Nationality] → 0_IsBadBuy lift 103.589 conf 90.8492
- [Covers_WheelType, MANHEIM_Auction] → 0_IsBadBuy lift 103.466 conf 90.7409
- [Covers_WheelType, AMERICAN_Nationality] → 0_IsBadBuy lift 104.983 conf 92.0714
- [Covers_WheelType] → 0_IsBadBuy lift 104.851 conf 91.9555

Some of these rules can be used to replace missing values. For example, if we want to replace the missing values of the variable *Nationality*, we can use an association rule similar to this one:

> [CHRYSLER_Make, 0_IsBadBuy]
> → AMERICAN_Nationality

With this association rule it is possible to say that the Chrysler is an American make. In our model we have used this kind of rule to replace all the missing values of the variables, *Nationality* and *Size*. In order to get all the rules to be able to replace the missing values, we have decreased the minimum support and then we have filtered only the needed rules. There are other interesting rules as this one:

> [LARGE_Size] →
> AMERICAN_Nationality lift
> 115.887 conf 96.904

Indeed, with a confidence equal to 96.9%, most of the *Large_Size* car in the data set are American. In this case the association rule, gives us a lot of information in comparison to a frequent pattern. This is due to the confidence measure that allows us to say that when we have a *Large_Size* car in the data set, we have a probability of 0.969 that this car is an American one. Without the confidence measure we are not able to make this kind of reasoning.

The associations rules we have extracted, can also be used to predict if a car is or is not a good buy. The important thing is to take into consideration the disparity of good buy in respect to the bad ones in the data set. For this reason, the association rules that have in the right inside *0_IsBadBuy*, should have a confidence greater than 90%; Since the number of good buy in the data set is the 87% of the entire data set. In the association rules that we have extracted, there are some examples that we can use for this purpose, like the following one:

> [2_VehicleAge,
> AMERICAN_Nationality] →
> 0_IsBadBuy (conf 93.7357)

Indeed this rule tells us that if a car has two years and it has an American nationality, with high probability it is a good buy.

We can now extract rules that have *1_IsBadBuy* in the right inside; To do that, we have decided to decrease the minimum confidence to 16%, because the number of bad buy in the training data set are the 13%. Additionally, for space reasons, we have filtered and reported only the rules that have in the right inside *1_IsBadBuy*

- [7_VehicleAge, AMERICAN_Nationality] → 1_IsBadBuy lift 177.591 conf 21.8414
- [7_VehicleAge] → 1_IsBadBuy lift 175.884 conf 21.6315
- [6_VehicleAge, AMERICAN_Nationality] → 1_IsBadBuy lift 147.423 conf 18.1312
- [6_VehicleAge] → 1_IsBadBuy lift 146.664 conf 18.0379
- [FORD_Make, 1000.0_WarrantyCost, AMERICAN_Nationality] → 1_IsBadBuy lift 142.158 conf 17.4837
- [FORD_Make, 1000.0_WarrantyCost] → 1_IsBadBuy lift 142.158 conf 17.4837
- [6_VehicleAge, MANHEIM_Auction] → 1_IsBadBuy lift 134.666 conf 16.5622
- [6_VehicleAge, MANHEIM_Auction, AMERICAN_Nationality] → 1_IsBadBuy lift 136.308 conf 16.7642
- [COMPACT_Size, AMERICAN_Nationality] → 1_IsBadBuy lift 135.257 conf 16.635
- [90000.0_Odometer, AMERICAN_Nationality] → 1_IsBadBuy lift 146.271 conf 17.9895
- [90000.0_Odometer] → 1_IsBadBuy lift 145.927 conf 17.9472

Now that we have these rules, we can build a model to predict if a car is or is not a bad buy. This can be done taking all the rules, which have in the right inside *1_IsBadBuy*. To evaluate a record, at this point, we simply check if this record has a match with the left inside of these rules. If we do have a match we can say that the record is a bad buy otherwise we can say that it is a good buy.

The model can be improved trying to get more association rules, that have in the right inside *1_IsBadBuy*. To get more association rules, we have to decrease the minimum support and using the same minimum confidence as the one used before.

In this section we have showed how to use this association rules to replace missing values and how to predict if a car is a bad buy or not. We have not tried to submit the model described in this section to Kaggle, because we can imagine that the model done in the classification part is way more accurate than this one.

Classification

As for the clustering part, before starting we had to convert non-numeric data into numeric data. To do that, we simply associate to each non-numeric variable a numeric number. Thanks to this mapping we are now ready to build our model using also the non-numeric data. After the previous analysis, in particular the data understanding and clustering part, we have decided that our model would have the following variables:

- *IsBadBuy*, this variable is needed in order to evaluate our model
- *VehicleAge*
- *VehOdo*
- *Auction*
- *Make*
- *WheelType*
- *Nationality*
- *VehBCost*
- *VNST*
- *WarrantyCost*

We have decided to build two different models: one using the given training data set without making any modifications and one balancing the number of good buy in comparison to the bad ones. This because if we do not do any balancing, the algorithm used to build our model, in case of nodes where there are a similar amount of bad buy and good buy, will probably say that that particular node is a good buy one, due to the unbalancing of the data set. After we have presented our models, we will choose the best parameters and algorithms to build these models and then submit both to Kaggle and show the results.

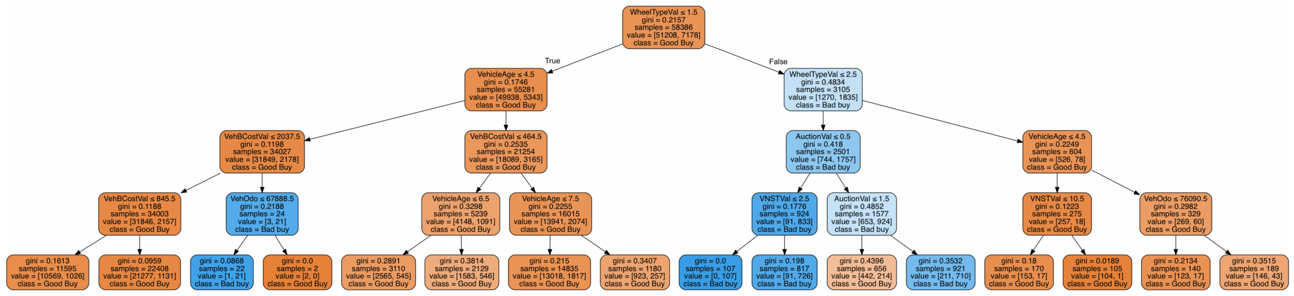

WheelTypeVal ≤ 1.5 gini = 0.2157 samples = 58386 value = [51208, 7178] class = Good Buy
True
False
VehicleAge ≤ 4.5 gini = 0.1746 samples = 55281 value = [49938, 5343] class = Good Buy
WheelTypeVal ≤ 2.5 gini = 0.4834 samples = 3105 value = [1270, 1835] class = Bad buy
VehBCostVal ≤ 2037.5 gini = 0.1198 samples = 34027 value = [31849, 2178] class = Good Buy
VehBCostVal ≤ 464.5 gini = 0.2535 samples = 21254 value = [18089, 3165] class = Good Buy
AuctionVal ≤ 0.5 gini = 0.418 samples = 2501 value = [744, 1757] class = Bad buy
VehicleAge ≤ 4.5 gini = 0.2249 samples = 604 value = [526, 78] class = Good Buy
VehBCostVal ≤ 845.5 gini = 0.1188 samples = 34003 value = [31846, 2157] class = Good Buy
VehOdo ≤ 67888.5 gini = 0.2188 samples = 24 value = [3, 21] class = Bad buy
VehicleAge ≤ 6.5 gini = 0.3298 samples = 5239 value = [4148, 1091] class = Good Buy
VehicleAge ≤ 7.5 gini = 0.2255 samples = 16015 value = [13941, 2074] class = Good Buy
VNSTVal ≤ 2.5 gini = 0.1776 samples = 924 value = [91, 833] class = Bad buy
AuctionVal ≤ 1.5 gini = 0.4852 samples = 1577 value = [653, 924] class = Bad buy
VNSTVal ≤ 10.5 gini = 0.1223 samples = 275 value = [257, 18] class = Good Buy
VehOdo ≤ 76090.5 gini = 0.2982 samples = 329 value = [269, 60] class = Good Buy
gini = 0.1613 samples = 11595 value = [10569, 1026] class = Good Buy
gini = 0.0959 samples = 22408 value = [21277, 1131] class = Good Buy
gini = 0.0868 samples = 22 value = [1, 21] class = Bad buy
gini = 0.0 samples = 2 value = [2, 0] class = Good Buy
gini = 0.2891 samples = 3110 value = [2565, 545] class = Good Buy
gini = 0.3814 samples = 2129 value = [1583, 546] class = Good Buy
gini = 0.215 samples = 14835 value = [13018, 1817] class = Good Buy
gini = 0.3407 samples = 1180 value = [923, 257] class = Good Buy
gini = 0.0 samples = 107 value = [0, 107] class = Bad buy
gini = 0.198 samples = 817 value = [91, 726] class = Bad buy
gini = 0.4396 samples = 656 value = [442, 214] class = Good Buy
gini = 0.3532 samples = 921 value = [211, 710] class = Bad buy
gini = 0.18 samples = 170 value = [153, 17] class = Good Buy
gini = 0.0189 samples = 105 value = [104, 1] class = Good Buy
gini = 0.2134 samples = 140 value = [123, 17] class = Good Buy
gini = 0.3515 samples = 189 value = [146, 43] class = Good Buy

Figure 49: Decision Tree without balancing of the variable *IsBadBuy*

Unbalanced Model. To train our model we have first of all split our training data set in two parts: 80% will be used as a training set and the remaining 20% will be used as a test set. This is done in order to have the possibility to evaluate the various decision trees that we are going to build, indeed the test set given by Kaggle does not have the *IsBadBuy* variable. After we have done this split, we have generated a decision tree using as criterion gini and setting up the max depth of the tree to 4. We have decided to limit the depth of the tree to have the Figure 49 of the tree more readable. This simple decision tree get an accuracy of 0.899 on the training set and an accuracy of 0.9 on the test set. From now on, we are going to evaluate our models using the cross validation technique. In this case, this model has reached an accuracy score of 0.90 (+/- 0.03). In the following paragraph we are searching to get a better decision tree.

To get a better decision tree we have tried with the Random Forest classifier and with the KNeighbors classifier. The Random Forest evaluation with the cross validation technique, get the same accuracy that we obtained for the first model generated. Instead the KNeighbors get a worst result than the other two 0.87 (+/- 0.00).

Due the fact that we have not reached no improvement, we have tried to tune the parameters and to do that, we chose to use *RandomizedSearchCV* and *GridSearchCV* of the scikit learn library. These functions gave us the possibility to get the best estimators trying different parameters. In addition, after we ran this functions, we did not get any improvement, reaching always 0.90 (+/- 0.03) of accuracy.

This result was unexpected and can probably be explained with the nature of the data set, that is unbalanced. Furthermore if we change classifier or parameters, we always get the same accuracy; Probably because the nodes of our trees will always have a majority of good buy and do not change their predictions trying these improvements. We are going to present a balanced model in further details in the following sections, trying to get a better result. We have decided to take for this unbalanced model the first one we have generated because it is the one with less depth than the others and for this reason it is the most efficient one.

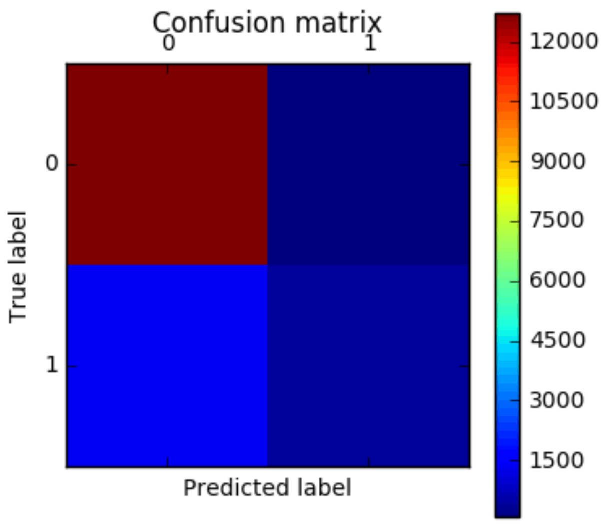


Figure 50: Confusion Matrix for the decision tree in figure 49

To evaluate our model we have reported the confusion matrix, in Figure 50, built with the test set. As we can see, the model makes a lot of mistakes predicting good buy cars that are actually bad buy, 1379 exactly. In contrary, it just made few mistakes, only 82, predicting bad buy cars that are instead good buy. We thought it is better to have a model that does not make errors, like the one we have built (i.e. false negative), rather than having a model that predicts bad buy when in contrary the car is a good buy. For example, if we let a buyer buy a car and tell him that it is a good opportunity, if then the buyer finds out that the car is a waste of money he will probably consider the model as a bad one. Instead, if we say to the buyer that the car is a bad buy but actually it is a good buy, he will not loose any money.

```
             precision    recall  f1-score   support

   Good Buy       0.90      0.99      0.95     12799
    Bad Buy       0.84      0.23      0.36      1798

avg / total       0.89      0.90      0.87     14597
```

Figure 51: Metrics for the model in figure 49

In Figure 51 we have some measures of the model. We have already said that the accuracy is 0.9 (+/- 0.3) and so the error is equal to 0.1 (+/- 0.3). We can see that our model is very accurate when it has to predict a good buy car, instead if the car is a bad buy it makes lots of errors.

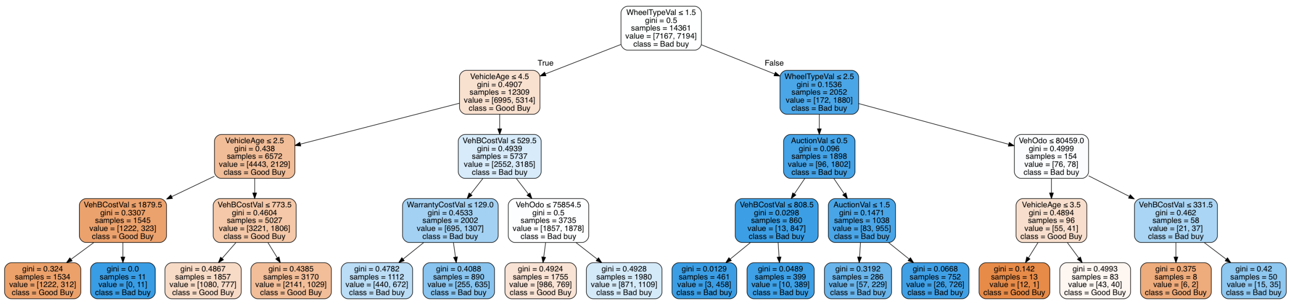


Figure 52: Decision Tree with balancing of the variable *IsBadBuy*

From the average measure the model seems to be very good, but if we consider an unweighted mean, the model will loose a lot of points. In the last section, where it is presented the score made by this model on Kaggle, we are going to see in details why this model is not so good.

Unbalanced model interpretation. The model has attended an accuracy of 0.9 (+/- 0.3) at the first try, and did not improved with the modifications we have explained in the last sections. This is due to the unbalancing of the data set as already explained. Another problem is the fact that the model does not perform well when the test set has a huge number of bad buy, and in contrary it works very well on a test set with a huge good buy. This is due to the fact that the model has been trained on an unbalanced data set. For these reasons in the next sections we are going to illustrate a model that has been built on a balanced training set.

Balanced Model. For this balanced model, we have divided the training set in good buys and bad buys. Then we removed records randomly for the good buys, until we reached a number of good buys equal to the number of bad buys. Afterwards, we have merged and shuffled the two data set and then split the result, our training set, in two parts as in the unbalanced model: 80% of training set and 20% of test set. Thanks to that process we now have a balanced data set. Since good buys are taken randomly, we have run this randomization multiple times to verify if the accuracy was similar between one randomization and another one. We have in fact tried with a simple tree as in the unbalanced case, and from one randomization to another one we have always reached an accuracy near to 0.65, this was a good indicator to understand that we can proceed with this model. One of the randomly generated tree can be found in Figure 52, and we can see that the number of bad buy nodes (the blue ones) are augmented in comparison to the decision tree in Figure 49, this is because we have balanced the training data set.

From the data measures, we have seen that the accuracy of this model is lower that the one of the unbalanced data set. In fact, as explained before, the unbalanced model is very good if the test set has a lot of good buy, whereas if we give the model a balanced test set it will not perform as good as it did in the previous paragraphs.

In Figure 53 we have the confusion matrix of the simple balanced model. Compared to the unbalanced model, we can see that we have in percentage more false negative and false positive. This because we have removed a lot of good buys and seen that the unbalanced model works well on unbalanced test set. We are not going to report the exact values of the confusion matrix, because the values change from one randomization to another one. From this confusion matrix we can see that there are: a majority of true negative, $\simeq$ 1300, the true positive are $\simeq$ 1100, the false negative are $\simeq$ 800 and $\simeq$ 450 of false positive. We are not going to compare these values with the unbalanced model, because, since the data set is now balanced it can lead us to wrong conclusion, but we are going to compare these values with a more accurate model that we are going to present in a couple of paragraphs.

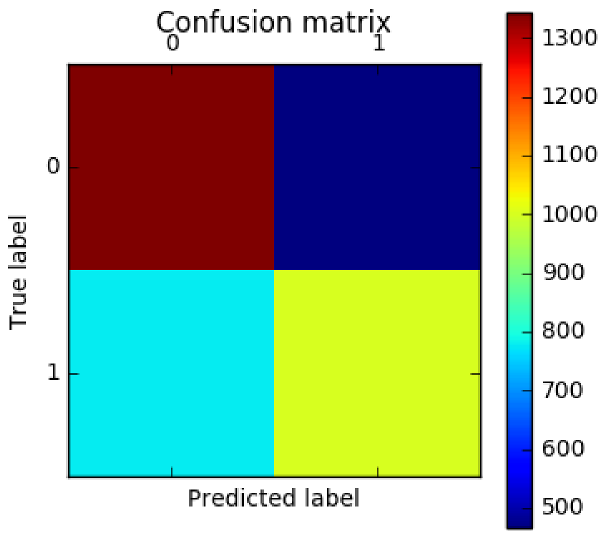
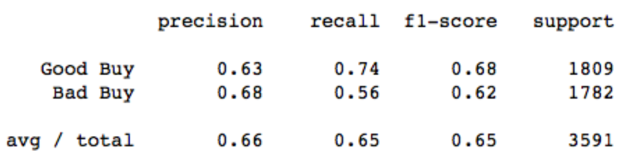


Figure 53: Confusion Matrix for the model in figure 52

This specific randomization got an accuracy of 0.64 (+/- 0.07). To calculate this accuracy, we have used the cross validation technique, as for the unbalanced model. From the measures in Figure 54 we can see that there is a big difference between the recall on good buy records (0.74) and the bad ones (0.56). This means that this model did not make lot of false negative errors, when the records given in input are good buys, but instead, if the records are bad buys it made lots of false negative errors. In the next paragraphs we are going to try to improve our model.



```
             precision    recall  f1-score   support

   Good Buy       0.63      0.74      0.68      1809
    Bad Buy       0.68      0.56      0.62      1782

avg / total       0.66      0.65      0.65      3591
```

Figure 54: Metrics for the model in figure 52

As for the unbalanced model, we have decided to try to improve our model. At first, we have tried to use the random forest classifier, obtaining a significant improvement in comparison to our base model, then, trying to use KNeighbors algorithm, we had a very bad accuracy, 0.55 (+/- 0.01). Thanks to this results we have decided to use the random forest classifier and try to boost the parameters to reach a good model. We tuned the parameters using, as for the unbalanced model, *RandomizedSearchCV* and *GridSearchCV*. These two functions, implemented by the scikit library, gave us the possibility to increase the accuracy of our model. The two functions gave us back two Random Forest with different parameters. The one coming from *GridSearchCV* function was the best, with an accuracy of 0.66 (+/- 0.05). Here listed the parameters used to boost the model:

- bootstrap=True
- class_weight=None
- criterion='gini'
- max_depth=4
- max_features='auto'
- max_leaf_nodes=None
- min_impurity_split=1e-07
- min_samples_leaf=1
- min_samples_split=2
- min_weight_fraction_leaf=0.0
- n_estimators=20
- n_jobs=1
- oob_score=False
- random_state=None
- verbose=0
- warm_start=False

The model we have used, uses gini as criterion to choose the best split, and it limits the depth of the tree to 4. It is also interesting to see that the function decided to set the parameters min_samples_leaf to 1 and min_samples_split to 2. Since this model is a random forest we have not found a way to represent it graphically, but we have some measures to show that this model is better than the base one (Figure 52).

Indeed, in Figure 55, we can see the confusion matrix obtained using this model. Comparing this matrix to the confusion matrix in Figure 53, we can see that in percentage there is a higher number of true positive and lower number of false negative. This is surely a good thing for this model. In order to have good values to compare the two models, these two confusion matrices were run on the same randomization of the data set.

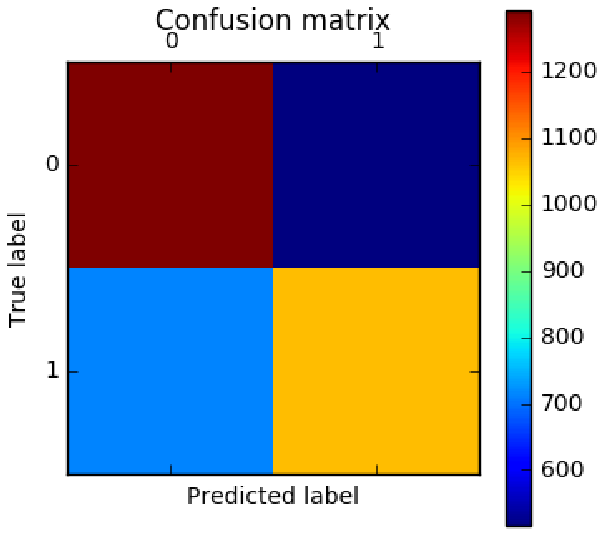
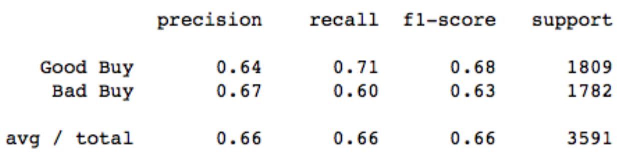


Figure 55: Confusion Matrix for the model

The confusion matrix in Figure 55 told us that the boosted model is the best one so far. Let's see if this is confirmed as well by the measures in Figure 56. In fact, in comparison to the measures of the base model (Figure 54), we have an improvement of the recall. Considering the bad buy records, the recall have passed from a value of 0.56 to 0.60. Instead, for the good buys we have passed from a value of 0.74 to a 0.71. So, in the end, we had an average improvement of only 0.01. As for the confusion matrices, also these measures have been calculated on the same randomized data set.



```
             precision    recall  f1-score   support

   Good Buy       0.64      0.71      0.68      1809
    Bad Buy       0.67      0.60      0.63      1782

avg / total       0.66      0.66      0.66      3591
```

Figure 56: Metrics for the model

Decision trees validation with test and training set. In this section we are going to report the results of our two models submitted to Kaggle. To do that, we have taken the training set and the test set given by Kaggle. We used the first set to build our model, using the same parameters that we have discussed in the previous sections, and then we ran the second set on the model generated, in order to submit a file csv to Kaggle. The first model we have submitted is the unbalanced model (Figure 49), and we obtained a score of 0.08040. After that, we submitted the random forest with boosted parameters, which uses a balanced training set, and got a score of 0.16976 reaching the 364 position in the rank of Kaggle.

Discussion of the best prediction model After the submission of our two models, we can say that the best prediction models is the balanced one, because it reached a score that is the double of the unbalanced model. As we have already explained in the previous sections, the unbalanced model did not work so well because, the test set given by Kaggle is non so unbalanced as the training set. additionally, we know that the unbalanced model does not work so well if the test set contains a lot of bad buy. The balanced model has obtained a good score because the training set in which it was trained was balanced.