POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Modeling Common Driver Behaviours on Vehicular Data: A Clustering Approach

Supervisors

Prof.ssa Elena BARALIS

Prof. Luca CAGLIERO

Dr. Giuseppe ATTANASIO

Candidate

Giulio BAGNOLI

April 2021

Abstract

Simplify and save: companies search for smart fleet management solutions with these two goals that enable them to manage their vehicles easily and efficiently. With this in mind, many tools have been developed, based on several indexes. Among them, driving behavior is one of the most common. It is mostly used to evaluate driver safety and to minimize fleet management costs.

In this thesis, a highly customizable framework has been proposed. Starting from the data provided by the devices mounted inside the vehicles (Controller Area Network), our tool can determine the driving behavior. To do this, several phases are performed whose outputs are effortlessly interpreted by a person not an expert in data analysis, as a fleet manager.

Particular attention was given to data cleaning, modeling and clustering. Supervised machine learning techniques (Random Forest and Gradient Boosting) have been applied to automatically classify vehicles into domain-specific classes. Since we want to group the trips based on the context in which they took place, we analyze the trips travelled on the same route. So knowledge takes on three levels of granularity during the different phases of the pipeline: GPS point, journey and route. The goal is to identify similar trips on the same route, to do this DBSCAN, a consolidated density-based clustering algorithm, is applied. It determines the driving behavior of drivers, highlighting any anomalies.

The experiments showed how anomalous trips within a route can be correctly highlighted by the developed software. The outputs given allow clear visualization of which trips are being analyzed and why they are assigned a certain driving behavior.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

While road freight traffic in Europe declined dramatically during the last year of the crisis, the strong prevalence of this method of transport over air, rail and river transport remains. This is largely because of the greater flexibility that can be ensured by road transportation for operators. Moreover, the characteristics of the territory and the lack of sufficient facilities force greater use of road transportation. Due to the high consumption of transport vehicles, trucking companies have to pay special attention to all the variables, that can influence this consumption, including driving behavior. It affects not only the financial sides of road travel but also the safety of drivers. Indeed, human responses and erratic driving style are the overriding causes of injuries. Building software that can track it, so that drivers can learn how to change, is important for these purposes.

In recent years, thanks to the possibility of collecting large amounts of data using technologies such as CAN Bus, various methods based on Machine Learning have been proposed for the study of driving behavior. The most common reasons are to detect violent actions or to build technologies that can support the driver when traveling. Regardless of the intent, the common solution is to use unsupervised machine learning algorithms - that is, models that can group descriptions of behavior based on their similarity. The widespread use of these techniques is due to the fact that it is very challenging to create datasets in which the actions of the driver are already classified. Indeed, it is not a trivial problem to define what the right driving behavior is. Similar behaviors can be both right and wrong in different contexts. Therefore, not only how the driver acts, but also the circumstances surrounding him should be included in the task of designing tools that can identify anomalous performance. Consequently, the purpose of this thesis is the construction of a highly customizable framework that, starting from the data provided by K-Master, can determine whether the behavior of a driver concerning a trip is correct or not. To do this, an innovative idea was exploited: behavioral research must be carried out not only on what the driver did but also on how other drivers behaved in a similar driving context. By analyzing the performance of multiple drivers in similar cases, driving behavior no longer depends on the context.

The data we operate on concerns the movements of K-Master's fleet in June 2020. The company offers two services to its customers: *Business* and *Connected*. The first service is the basic one, it retrieves standard information such as position, speed and date. The second one, in addition to retrieving basic information, also acquires more technical information regarding the state of the vehicle. Over time, the *Connected* service has been provided through different devices that use different protocols to communicate information, for this reason, the vehicle status data has different formats. It was therefore necessary to accomplish an in-depth exploration of the data provided by this type of service to understand which indicators were available and if their quantity was sufficient to use them.

When we work in a scenario with real data, they are often dirty or missing, for this reason, it is good practice to put a preprocessing step before the machine learning algorithms. So the developed framework was divided into two parts. In the first one, information is encoded to bring it to such a state that it can be easily interpreted by the algorithms, while in the second part the driver behavior is determined. In particular, in the first part of the pipeline, the data are clean and aggregated, we initially perform these cleaning operations as any incorrect data can compromise the result (garbage in, garbage out principle). Then each vehicle is optionally assigned its type using the K-Master's registry. Currently, the process of assigning the type of vehicles is generated by humans while with the introduction of our framework it is made automatic. To do this, various solutions have been proposed based both on some heuristics and on classification algorithms. Next, the trips are identified: these are set of consecutive GPS positions whose temporal distance is not greater than a certain threshold. In this way, we move from knowledge related to a GPS point to knowledge at the travel level. Subsequently, trips that start from the same area and arrive in another area in common are grouped, the path traveled by these vehicles is called route. The reason for this grouping is that the behaviors within the same route depend on the same context. In conclusion, routes were ordered according to their importance.

The last part of the pipeline determines the driving behavior of the trips that belong to a route. Within this, we do not know how many driver behaviors there may be or if there are anomalous ones among them. For this reason, the clustering task is carried out by DBSCAN (Density-Based Spatial Clustering of Applications with Noise) which is an established density-based clustering algorithm concerning a set of points in a neighborhood. This algorithm does not require to know in advance the number of clusters that will form, it creates as many clusters as there are “fairly” dense groups within the data. Another reason for using DBSCAN is that it can detect any anomalous behaviors and not put them in the nearest cluster, as K-Means does.

Since maintaining correct driving behavior is in the interest of both drivers and companies that manage fleets of vehicles, consider that most drivers assume correct, or at least responsible, performance seemed to us a good approximation. Bad behavior, on the other hand, can easily be identified as abnormal conduct as it is likely not commonly experienced. Therefore, in the experiments in which we assign driving behaviors, we considered the behavior of clusters correct, since this is the most common, and that of outliers incorrect, as they are anomalies. By analyzing their characteristics, it is possible to understand why these behaviors are considered anomalous and how they differ from the virtuous ones. In the last part of this phase, that does not concern this thesis, a KPI is assigned for each trip, this is an index of the driver’s skill on that trip.

The analysis performed to validate the automatic assignment of the vehicle type shows that the approach based on the presented heuristics is not sufficiently precise, on the contrary, the proposed classification algorithms, which are Random Forest and Gradient Boosting Tree, obtain promising results. The study of the CAN data shows that it is not possible to use the CAN features for most vehicles and that, for those few vehicles with a *Connected* service, only a minimal part of these features is available.

In the experiments in which we assign driving behaviors, two or more clusters are never formed, therefore the framework, with the parameters used, identifies at most two driving styles. The one of the cluster and the one of the outliers. The tests carried out show how the behaviors of the outliers almost always detach negatively compared to that of the centroid, confirming our hypothesis. In most cases, trips are considered outliers due to excessive speed or fuel consumption.

Future works can improve the accuracy of the analysis. A possible strategy could be to change the concept of route, defining it no longer based on a pair of cells but based on the path followed. In this way, we avoid reporting as anomalous a trip that followed a different road than the centroid of that route. Implementing that was impossible in this thesis as the scarcity of data would have led to routes containing too few trips for analysis. With the increase in the amount of data available, it would also be possible to apply Deep Learning strategies. Techniques that allow incremental training of the model certainly are useful in the latter case. It would also be interesting to study the application of other density-based clustering techniques capable of determining clusters with different densities as OPTICS and HDBSCAN.

This document is organized as follows. Chapter 2 introduces the problem of identifying driving behavior, explaining how this can be used in various fields. In particular, the emphasis is on how his study can help a fleet manager. Chapter 3 describes which techniques to use in the data pre-processing phase. Then one explains the algorithms used to identify the type of vehicle, highlighting the pros and cons. Finally, the problem of clustering is introduced, in particular, the functioning of the algorithm used is explained. Chapter 4 gives a brief account of the state of the art in identifying driving behavior. Chapter 5 lists the various data sources. The sixth chapter contains the description of the pipeline proposed to K-Master and some definitions necessary to fully understand it. The experiments were conducted and the results are reported in Chapter 7. Finally, in Chapter 8 the conclusions are presented.

Chapter 2

Driving Behavior Analysis

The concept of driver behavior has recently become an emerging subject of great importance. It is helpful in the automotive and smart transportation, automotive insurance and public transport sectors. The main aim is to understand the possible correlation between the behavior of the driver and the collected data. Assigning behavior class is a challenging topic given the stochastic nature of driving.

To develop creative solutions that enhance efficiency in previous fields, comprehension of driving style is important. The topic of driver behavior assignment has been related many times in the literature, including collecting navigation data using an on-board computer, tracking the physical state of the driver, etc. The purposes for which this task can be used are divided into three macro groups, following the grouping described in [1]:

- Vehicle-Oriented Application: The goal is to develop a system that can adapt to the environment in real-time, enhancing the driving experience. In this context, the solutions proposed focus on a single vehicle and vary according to the degree of autonomy. Intelligent vehicle systems and autonomous vehicles are the main applications [2, 3], these are recent fields of research aimed at automating the function of vehicle by driving behavior assignment. The automatic detection of accidents is another very significant application. The main function of this field is to determine whether or not the driver has had an accident and to automatically send help accordingly [4, 5].

- Management-Oriented Applications: The objective is to develop models that are capable of improving both infrastructures and fleet management. In the first case, the proposed solutions are crucial to monitor road state, it is possible to evaluate the integrity of a road given the driver behavior of those who travel it [6, 7]. While in the second case, the proposed solutions are adopted to monitor the resources used by drivers, reduce their risk of accidents and improve the efficiency of their services[8, 9]. The approach developed in this work falls into this category as the main objective is to characterize the driver behavior to manage the fleet.

- Driver-Oriented Applications: To guarantee a safer driving experience, the aim of these techniques is to monitor the operator. The proposed solutions concentrate on the driver and base on features relating not only to the state of the vehicle and the environment but also to its condition. Some of these characteristics can be heart rate or eye activity, which are used to track, for example, sleepiness or the degree of awareness. In this category, driver attention evaluation and distraction detection are the major topics [10, 11].

2.1 Fleet Management

Drivers are at the core of a company for fleet managers, from tackling speed to maintaining vehicles, driver behavior is crucial. Using a proper model for driver style assessment, it is possible to assign a score to the operators before they become a significant complication, in this way we can highlight any issues of individual drivers.

Although the training of drivers, on the one hand, is certainly an important method for avoiding critical problems during their activities, on the other hand, the fleet manager can not be there to see how they act. While in the past, a bad driver could ride for months, risking the image of a business and causing needless wear and tear to costly vehicles, nowadays it can be avoided. Manual tools have been used for the gathering of feedback in recent decades, seldom leading to any intervention due to the slowness of the information collected and the lack of techniques capable of carrying out an in-depth study of the data collected.

The main aspects that are influenced by the driver's behavior are:

- *Fuel Consumption.* It is a fact that the style of driving has a significant influence on this consumption [12, 13, 14]. Dedicated research on the link between driving behavior and consumption of fuel will help minimize transport energy costs. Certain activities can harm fuel use and efficiency, such as sudden acceleration, hard braking or idle time.
- *Vehicle Wear.* Incorrect actions, such as hard braking and fast acceleration, have a significant long-term effect on vehicle performance [15, 16]. We can both correct these habits by tracking driver data in real-time, and predict when vehicles need to be changed before they are to be scrapped.
- *Risk of Accident.* Accidents, particularly on long and tiring journeys, can happen while driving, but that does not mean we can not do our best to avoid

> them. In road safety, driver behavior plays a dominant role [16, 17]. With a fleet management solution, a fleet manager can quickly recognize which of their drivers are not meeting their safety requirements and introduce additional training programs and rewards to keep their company safe.

Through smart fleet management, it is possible both to monitor the actions of the individual driver and to have a higher perspective, evaluating the different activities of multiple drivers on a single route.

Chapter 3

Theoretical Background

The challenge is to develop and implement a pipeline, which starting from GPS and CAN data, may group the behavior of drivers, highlighting any outliers. The knowledge derived from this grouping will be used to assign a score to each trip and offer the services mentioned in section 2.1.

Two macro-steps need to be addressed: data preprocessing and clustering. The first phase is composed of a set of data mining techniques that transform raw data into a more understandable, useful and efficient format. While in the second one groups the data based on its similarity with DBSCAN. In the work carried out, to offer a more specific service, during the preprocessing part, an optional script is added. It classifies the vehicles according to their type, dividing them into HEAVY and LIGHT. A brief explanation of the algorithms used to make classification is given in the section 3.2.

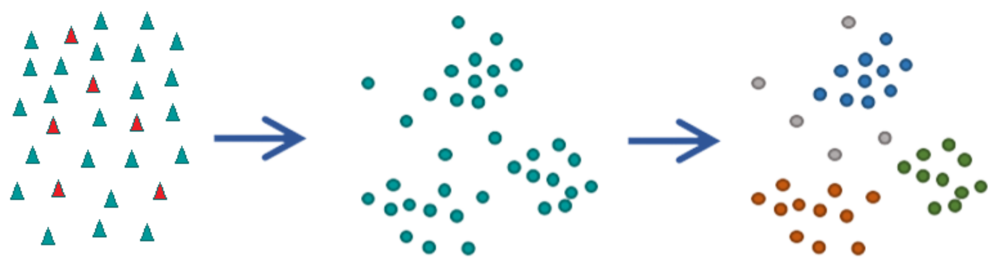


Figure 3.1: Graphic representation of the two main phases. In the first phase, we perform cleaning operations, bringing the data into a form understandable by the algorithms. In the second phase, we create the clusters, each of them represents a different driving behavior.

3.1 Data Peprocessing

When we work in a scenario with real data, they are often dirty or missing. For this reason, it is good practice to place a data preprocessing step before the machine learning algorithms. Data is converted or encoded in this process to bring it to such a state that it can be easily interpreted by the algorithms.

The most common problems related to the lack of correct preprocessing are listed below.

- Missing data is the most common problem. Data can be missing in two ways: random and non-random. If information is manually entered and someone has forgotten to insert it, the first case occurs. Otherwise, if the lack of data is due to the value of it, the second case occurs.
- Inconsistency of data is an important concern because it produces data anomalies that actually shouldn't exist. A simple example of data inconsistency is when someone starts to distinguish "cars" from "trucks". Then, instead of using the full word, changes coding and starts calling cars "C" and trucks "T". Another case of inconsistency is when different information is stored in the same variable.
- Missing anonymization occurs when part of the data is not removed before running machine learning algorithms. The reasons why data removal should occur are related to privacy, confidentiality, security and ethics. Failure to anonymize data could be a source of bias.
- Outliers strongly affect the performance of a machine learning model. It is very difficult to classify them since they may also be the particular result of a data gathering error or a particular phenomenon that should be taken into account for data processing.

Several strategies have been developed to address these and other less common problems.

3.1.1 Data Cleaning

Data cleaning techniques aim to remove errors and inconsistencies, improving data quality. The philosophy behind these approaches is to transform dirty data into clean one.

- **Missing values.** It is one of the most frequent issues solved by data cleaning tools. Missing values can not be ignored in a data set, especially because they can not be accepted by many machine learning algorithms. There are several mechanisms to deal with the problem, the choice of which method to use depends on the specific problem domain. A data scientist can fill the missing data manually, a very time-consuming method, or fill them with a constant that can be "0" or a mean value.

- **Noise and outliers.** Binning can be used to control noise. In this technique, the sorted data is loaded into containers. It is possible to build containers starting at an equal distance or at an equal frequency. Through bin mean, bin median, or bin limits, attenuation may occur. Another strategy for eliminating noisy values is to identify thresholds and remove all values which do not fall within them. Only if we have the domain information required to define these thresholds this approach can b used. Besides, it is possible to use graphical tools to identify the outliers, if the size of the data allows it, or clustering techniques, better described in section 3.3.

- **Unwanted data.** They can be obsolete or duplicated data, which may be present in situations where values have been aggregated from multiple sources. In addition to taking up unnecessary space and slowing the creation of models, redundant information can lead to a worsening of the results obtained.

3.1.2 Data Reduction

They are techniques used in those situations where the data's dimensionality is high. Their objective is to reduce it by choosing which characteristics to exclude or merge. These operations can be done either automatically, by algorithms that approximate the importance of the features within the dataset, such as the PCA, or by domain experts who can understand the data analyzed. The problem of "Curse Dimensionality" can be avoided by data reduction operations.

Curse of Dimensionality If the data dimensionality grows we have fewer samples per region, figure 3.2 [1], the nearest neighbours might be very distant from the point under analysis and therefore they are very different from the latter. Another negative aspect of the dimensionality increase is that, while the neighbours are far from our point, the other points are very near to the neighbours.

So when we are in high dimensionality the concept of *closeness* becomes less and less meaningful, the assumption that neighboring points are similar to each other is broken because neighbours are not particularly closer (and therefore similar)

[1] https://deepai.org/machine-learning-glossary-and-terms/curse-of-dimensionality

than any other points in the dataset. The better thing to do in this condition is to reduce the dimensionality.

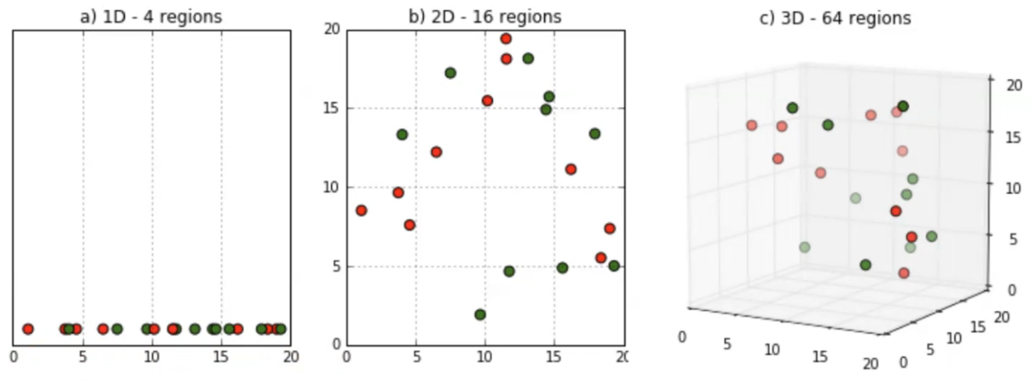


Figure 3.2: Representation of the Curse of Dimensionality. In the figure we see how the number of examples per region decreases as the number of dimensions increases.

3.1.3 Data Aggregation

Any process in which information is collected and expressed in a synthetic form is data aggregation. When the data is aggregated, the rows of atomic data are replaced with summary statistics on those observations. In a data warehouse, aggregated data is usually found. It can respond to analytical questions and thus significantly minimize the time taken for massive data sets to be queried. Data aggregation is often used to provide a useful overview of data when it is too detailed. E.g. if we want to analyze the behavior of a driver on a route we must first summarize the information of the GPS points for each trip on that route and then summarize the trip information on the route. In this way, it is possible both to display the information of hundreds/thousands of GPS points in a few lines and easily understand this information.

So the data aggregators summarize the statistics from many sources, among these aggregators the ones used in this work are: the minimum, the maximum, the average, the variance. Aggregated data does not have to be numeric, e.g. count the number of data rows.

Before aggregating, it is necessary to verify both the accuracy of the data to be aggregated and that these are in sufficient numbers. Besides, it is essential to perform the strategies presented in section 3.1.1 before the aggregation phase, as the inclusion of dirty data can jeopardize the aggregation.

3.1.4 Data Normalization

Without collapsing gaps in the ranges of data values, all these processes attempt to modify the numerical features in the dataset to a popular scale. Before moving on to the data analysis, it is a great idea to always normalize the data. In particular, normalization must be performed when the various characteristics in the data have various scales and the algorithms use the concepts of distance or closeness. Some of the most common approaches to data normalization may be: Min-Max, Z-Score, to name a few.

3.2 Classification

Classification is the task of predicting the label of given points, basing it on a set of points whose classes are known [18, 19]. The classification is therefore a type of supervised learning. Predictive models are approximations of functions that map the space where the samples live in the discrete space of the classes. An algorithm that performs the classification task is known as a *classifier* or *learner* [20, 21]. There are two categories of learners: *lazy* and *eager*.

- A lazy classifier contains all the training set points and acquires their information only in the test phase. In those situations where the training set is highly dynamic, this type of classifier is especially useful.

- An eager classifier builds a predictive model in the training phase before moving to test one. If on one hand, the latter is much faster than that of lazy learners, on the other, the training phase can be very long.

Many models perform the classification task, the choice on which use must be based on the data available. If the samples are readily classifiable, complicated models should not be used (e.g. linearly separable). The models vary according to a set of properties that can be summarized in:

- **correctness**: shows how good the learner is at classifying. There are various measures to quantify it based on the number of samples classified correctly or not. Among these, *precision* and *recall* are often used to verify how accurate the predictions on individual classes are. The first is calculated as the ratio of the number of elements correctly classified for a class to the total of elements classified for that class. While the second is calculated as the ratio of correctly classified elements for a class to the total number of instances of that class.

In mathematical terms, it is possible to express them as:

$$Precision = \frac{True\ \ Positive}{True\ \ Positive + False\ \ Positive}$$

$$Recall = \frac{True\ \ Positive}{True\ \ Positive + False\ \ Negative}$$

- **time**: refers to the computational cost to train the model and to perform the classification. As anticipated before, there are both models whose training phase is very short but very slow to carry out the classification (eg KNN) and models that are slow in training and relatively fast to classify (e.g. neural networks).
- **strength**: is the model's ability to correctly classify an example even if some of its features are missing or wrong.
- **data size**: refers to the ability of the classifier to be independent of the size of the dataset. That is, the performance of the learner should not vary as the size of the dataset changes.

3.2.1 Decision Tree

It is one of the simplest algorithms used to solve classification problems [22, 23]. By learning the decision rules derived from the training set, the objective is to create a tree that can estimate the value of a label. To do this, the data is continuously divided with respect to one of their feature.

Three components make up a tree: nodes, branches, and leaf nodes. In the first, data is split on a feature to a condition. The name of the first node is *root*. The branches correspond to a node's output, connecting a node to another node or to a leaf node. Only one incoming branch per node can be present. Leaf nodes are the terminal ones, inside them are contained the previsions, which represent the label.

The Gini score is usually calculated to select which feature to use to split, this amount shows how good a split is based on how mixed the classes are in the split-generated groups. If the data dimensionality is very high, stop criteria can be used to avoid building a tree that has as many levels as there are data features.

Pros

From figure 3.3 it is possible to note that the models created are easily interpretable. The algorithm allows developers to confirm that knowledge from the data has been

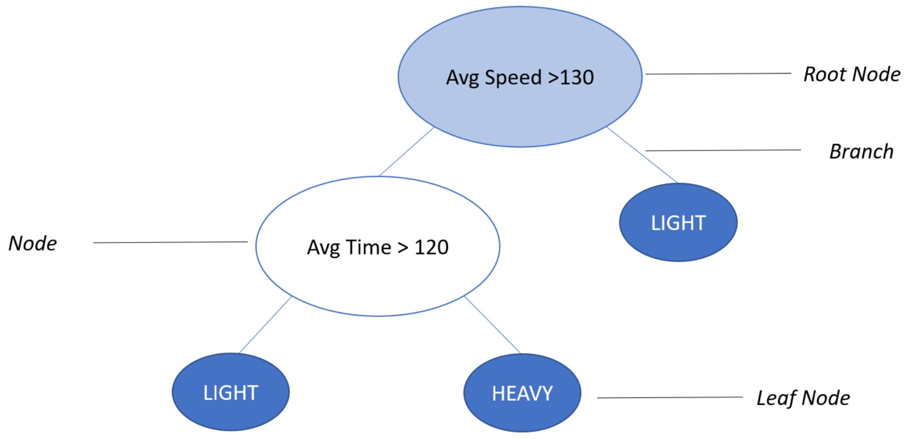


Figure 3.3: Example of a Possible Decision Tree.

gathered from the learner and enables end-users to have trust in the model's decisions. Furthermore, the cost of using the tree is logarithmic in the number of data points used to train the model and the learner can manage both categorical and numerical features.

Cons

By learning the decision rules derived from the training set the algorithm can lead to the construction of models not able to generalize on new data (overfitting problem), to counteract this it is possible to use some mechanisms such as pruning and its variants [24, 25]. The algorithm that chooses which feature to split and which condition to use is greedy therefore cannot guarantee a globally optimal solution. This can be mitigated by using decision tree-based techniques where many trees are trained (e.g. Random Forest). Another drawback to decision trees is that they do not support missing values

3.2.2 Random Forest

The concept behind the random forest is to create many decision trees and use them all when we have to predict a label, so it is an ensemble learning method [26, 27]. Each tree votes for a class, the predicted label is the one with the major number of votes. Trees are built on a random subset of the training set to not have all the same of ones within the forest. When we have to choose the feature on which the samples are split, this is selected from a random subset of the characteristics.

Pros

Random forest works very well for large datasets by constructing its trees with a sub-set of the training set. Similarly, on high-dimensional inputs, it also acts well. In addition, since not all the available features are chosen for the feature's split, the algorithm can not be considered totally greedy and is, therefore, more likely to achieve an optimal solution than using a single tree. The random forest algorithm always suffers less from the overfitting problem than a single decision tree, but it still suffers from it.

Cons

While random forests often achieve higher precision than a single decision tree, the intrinsic interpretability inherent from it is sacrificed. Since they are more complex and therefore require greater computational workloads, training a random forest is slower than training a single tree,

3.2.3 Gradient Boosting Decision Tree

GBDT improves the accuracy of a tree by minimizing a generic cost function, contrary to the approach used for random forests, where a model's performance has been improved by introducing additional models. The GBDT uses the boosting algorithm [28], from which it inherits a strong learner's construction logic. This strategy iteratively constructs a series of decision trees. Each one is trained and pruned on examples that have been filtered by previously trained models. Examples that have been incorrectly classified by the previous learner in the ensemble are resampled with a higher probability to give a new probability distribution for the next tree in the ensemble to train on [29].

Nowadays there are faster versions of the boosting algorithm reported in [28] specific for decision trees, among these one of the most interesting is the one developed by G. Ke *et al.* in [30].

3.3 Clustering

The principal activity of exploratory data processing and data mining applications is clustering. Clustering is the task of grouping a set of data in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups (clusters)[31]. Within a large data volume, this method helped us to identify the most crucial information. This is one of the most significant issues in the era of Big Data. The task of the cluster is subjective and samples can be grouped by several aspects.

In terms of intra-cluster similarity and inter-cluster similarity, the performance of the generated clusters can be calculated. A correct algorithm produces groups where the first value is high and the second one is low. These values may depend both on the choice of the similarity measure and the algorithm used. In addition, a good algorithm for clustering is not susceptible to noise and outliers.

A clustering algorithm should be capable of:

- to scale well over large amounts of data.
- to handle the possibility of having attributes of different types.
- to determine clusters of any shape.
- to manage noise to improve the quality of the clusters.
- to handle data with high dimensionality, if they are not able to do so we can always use the techniques seen in section 3.1.1.
- to generate results interpretable and usable.

Furthermore, a useful algorithm should have the minimum number of input parameters, since their choice is always complicated and can usually lead to introducing bias into the results. The last property to consider is that the result should be independent of the order in which the data are analyzed.

It is possible to classify the clustering algorithms in various ways, a first division can be made by considering how these techniques generate clusters:

- bottom-up: each sample is contained within its cluster, recursively similar clusters are merged into a single group.
- top-down: all samples are placed in a single cluster which is iteratively divided into smaller groups.

Both approaches stop when a specific condition is reached, such as the formation of a desired number of clusters. In the figures 3.4 and 3.5 there is a graphical schematization of the two processes. Another important subdivision can be made between *Hard* and *Soft* algorithms. If we use a hard strategy each sample belongs to one cluster at the end of its execution. While a probability of belonging to each cluster is assigned for each sample when we use a soft algorithm. The most popular types of Hard Clustering Algorithms are Hierarchical and Partitioning.

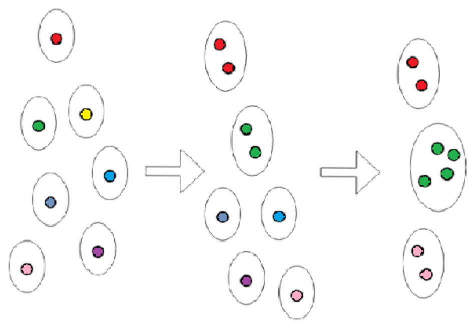


Figure 3.4: Bottom-up representation.

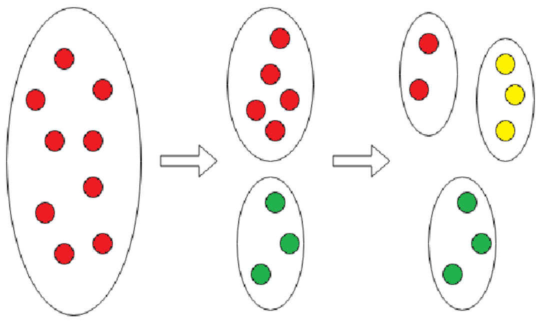


Figure 3.5: Top-down representation.

3.3.1 Hierarchical Clustering

The hierarchical clustering approach merges or splits similar data by building a hierarchy tree of clusters also known as dendrogram [32]. Each cluster set has a different height that shows how well the clusters are separated. The formation of clusters is progressive, if the approach with which the clusters are created is bottom-up then the algorithm is called agglomerative while if the approach is top-down the algorithm is called divisive. In the figure 3.6 there is a graphic schematization of the two processes.

The pros of this approach are that we do not have to choose the number of clusters a priori and it is easy to interpretable [33]. Since once the cluster has been formed the node it cannot be revisited [34] we can not reach a global optimization. There is no iterative approach, typical of Partitional Clusters. However, the studies conducted in [32] show how this type of clustering acts accurately and quickly on small datasets whose samples have a small dimensionality.

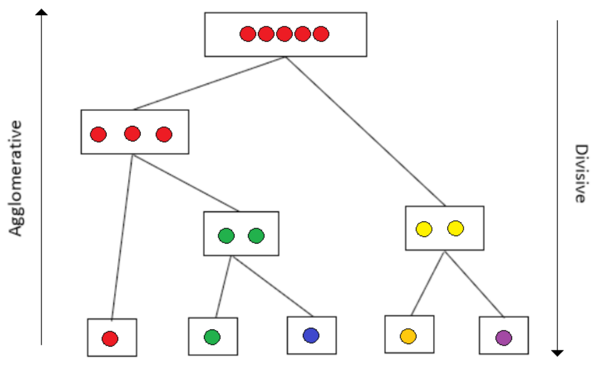


Figure 3.6: Dendrogram Structure for Hierarchical Clustering.

3.3.2 Partitional Clustering

Since these techniques are faster than hierarchical algorithms, they are the most widely used in real cases where the cardinality of the dataset is a critical aspect [35].

Defining the similarity of samples based only on their proximity can be limiting. Partitional algorithms can be classified based on the definition of similarity they use. They fall into four categories: Centroid based, Density based, Model based and Graph based. An example of the different results of the various methods is shown in figure 3.7 [2]

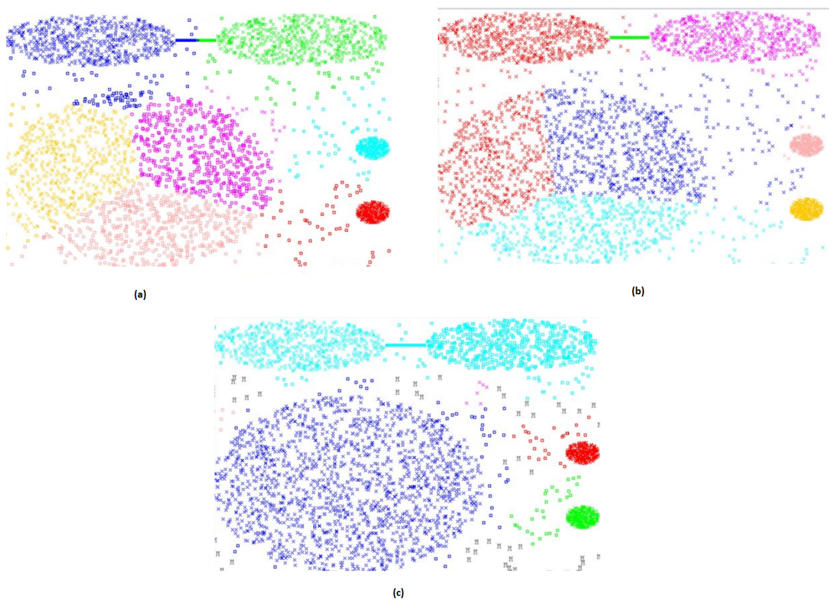


Figure 3.7: Graphical representation of different Partitional Clustering Algorithm. (a) Centroid Based Clustering; (b) Model Based Clustering; (c) Density Based Clustering

Centroid Based Clustering : these are iterative algorithms for clustering in which the notion of similarity is extracted from the proximity of a data point to the cluster center. The centroids are chosen based on criteria. Usually, the sum of square error is used as a measure to define a point as a centroid. We speak of meioids if the points, which are identified as cluster centroids, do not belong to the dataset.

[2]The image is inspired by those present in [36]

In this group, K-mean is the most used algorithm since it is simple to implement and interpret. The final cluster number must be known a priori in these methods and, for this reason, they are typically used in those cases in which this information is known. The other negative elements of K-means are that it does not always converge to the global optimum and that it is susceptible to outliers and noise. By introducing simulated annealing techniques, the issue of the sub-optimality of the solution can be solved.

Model Based Clustering : these are based on a probabilistic approach. Their basic philosophy is that all samples belong to probability distributions. Clusters are formed according to the point's belonging distribution. The two most popular ways to do this are classification likelihood, which maximizes parameters estimate, and Mixture likelihood, which assumes probability function as a sum of weighted component densities.

Density Based Clusterings : these methods that identify distinctive clusters in the data, based on the idea that a cluster in data space is a contiguous region of high point density, separated from other such clusters by contiguous regions of low point density. The data points in the separating regions of low point density are typically considered noise/outliers [37]. Using density concept these kinds of strategies can discover clusters of arbitrary shapes [38].

The most common algorithm is DBSCAN, contrary to K-mean, it does not need the apriori knowledge of cluster numbers. It is based on two parameters which are the maximum distance for two points to be considered close and the minimum number of neighboring points to form a cluster. The optimal value of these parameters can be estimated with various techniques such as knee research. The downsides of the algorithm are that it is not completely deterministic and that the choice of distance measurement can impact the quality of the result.

Other common density-based algorithms are OPTICS [39], HDBSCAN [40] and Border Peeling Cluster [41]. The first aims to create an augmented ordering of the database representing its density-based clustering structure instead of explicitly producing a set of clusters. The work explained in [40] elaborates a clustering hierarchy from which a simplified tree of significant clusters can be constructed. Finally, the Border-Peeling algorithm, by iteratively identifying the border points of the clusters, is able to separate the various groups.

Graph Based Clustering : these algorithms are based on Graph theory. Samples are considered nodes and a link is established between them through graph or hypergraph. The maximum connected sub-graph is equivalent to single link clustering in hierarchical clustering, while the maximum complete sub-graph corresponds to complete link clustering in hierarchical clustering [32]. We can also use graph theory combined with probability as happens in the CAST algorithm [42].

3.3.3 DBSCAN

In 1996, Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu proposed for the first time in [43] the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm. The goals behind this implementation were:

- the absence of parameters in the algorithm, such as the final cluster number. Determining a priori the optimal value of any algorithm parameters requires a minimum of domain knowledge which is not always available.
- the possibility of discovering clusters of any shape, this is an important feature that is not always present in the most common algorithms. The K-mean algorithm, based only on the concept of distance, recognizes only spherical clusters.
- good efficiency on large databases.

As mentioned in section 3.3.2 DBSCAN is part of the density based clustering algorithms. The rest of the section is organized as follows: preliminary knowledge, algorithm, parameter estimation, advantages and disadvantages.

Preliminary Knowledge

The basic rule of the algorithm is that a point belongs to a cluster if it is close enough at least a minimum number of points in the cluster. From this assumption, we can immediately understand that the algorithm is based on the concept of density of points in space. To define the density we need two parameters which are:

- ϵ represents the maximum distance that can exist between two points that are considered close.
- minPoints is the minimum number of points to define a cluster.

BDSCAN counts the number of points contained in the neighborhood of each point and classifies the sample based on this value. By neighborhood, we mean that region of space contained in a hypersphere of radius ϵ centered at the point. A point can be characterized in the following way:

- *Core point*, if there are at least minPoints points within its neighborhood (including itself). Each cluster has at least one core point.

- *Border point*, if within its neighborhood there are no at least minPoints points but it is contained within a neighborhood of a *core point*. These are the points that determine the boundary of the cluster.
- *Noise point*, all other points. These do not belong to any cluster and are therefore considered outliers.

To understand the algorithm three other definitions must be introduced: *directly density-reachable*, *density-reachable* and *density-connected* [43].

Directly Density-Reachable A point q is said to be directly density-reachable by a core point p if it is found within its neighborhood.

Directly density-reachability is symmetrical for pairs of core points but it is not in general.

Density-Reachable A point p is density-reachable from a point q w.r.t ϵ and minPoints if there is a chain of points $p_1 \dots p_n$, $p_1 = q$, $p_n = p$ such that p_{i+1} is directly density-reachable from p_i.

Density-reachability is an explicit extension of directly density-reachable and such as this is not symmetric, however, density-reachable is transitive.

Density-Connected A point p is density-connected to a point q w.r.t. ϵ and minPoints if there is a point o such that both, p and q are density-reachable from o w.r.t. ϵ and minPoints.

Density-connectivity is a symmetric relation. With this, we can define a density-based notion of a cluster. Intuitively, a cluster is defined to be a set of density-connected points which is maximal w.r.t. density-reachability [43].

In figure 3.8, minPoints= 4. Point C is the only core point because its neighbor contains 4 points (including the point itself). Points B, E and D are not core points, but are reachable from C and thus belong to the cluster as well. Point A is a noise point as it is not contained in the neighborhood of a core point.

Algorithm

Conceptually, the algorithm can be summarized in the following steps [44]:

1. For each point the number of points contained in its neighborhood is counted and the core points are identified.

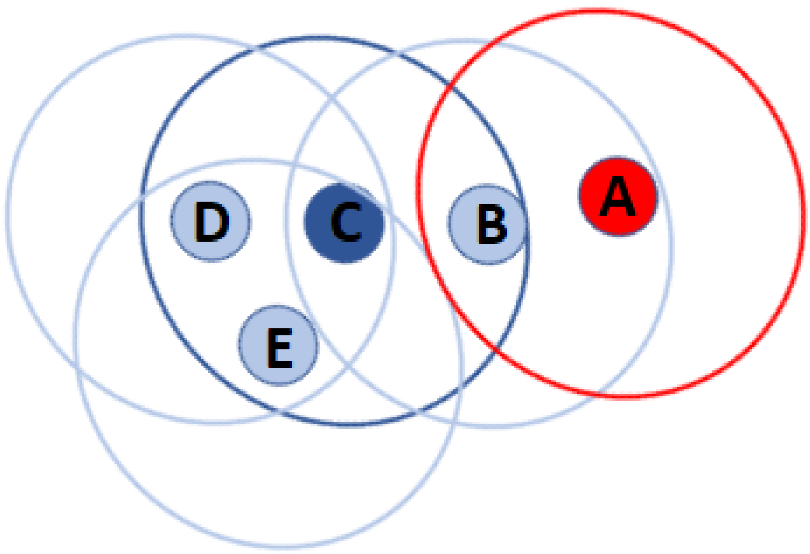


Figure 3.8: Example of assigning the type to the various points.

2. The connected components of the core points of the neighbor's graph are calculated. The neighbor's graph is a graph in which each core point is a node, two core points are connected by an arc if their distance is lower than ϵ.
3. The points contained within the neighborhoods of the core points of a single connected component are merged in a cluster. If a point does not belong to a neighborhood of a core point it is an outlier.

From a formal point of view the algorithm starts from a random point p in the dataset and retrieves all density-reachable points from p w.r.t. ϵ and minPoints. If p is a core point, that is, if at least minPoints points have been recovered, then a cluster is generated, and the points recovered by p are analyzed. These points belong to the cluster of p. If between these points there is a core point q then the elements contained in the neighborhood of p and q will form a single cluster. Once all the points of the cluster have been visited or if p is not a core point, we proceed with a new random point not yet visited by the algorithm. The pseudo-code of the algorithm is shown in algorithm 1.

As we can see in algorithm 1 if a point is initially identified as an outlier and it is density-reachable from any unexplored point of the dataset, it could still become a boundary point of a cluster. As anticipated in paragraph 3.3.2 there may be situations in which the algorithm is not deterministic and the result depends on the order of analysis of the points. This happens when there are two clusters,

Algorithm 1 DBSCAN algorithm. D represents the dataset and Minpts represents minPoints.

```
procedure DBSCAN(D, $\epsilon$, Minpts)
    $C = 0$
    for each unvisited point P in dataset D do
        mark P as visited
        NeighborsPts = regionQuery(P, $\epsilon$)
        if sizeof(NeighborsPts) < Minpts then
            mark P as NOISE
        else
            C = next cluster
            expandCluster(P, NeighborsPts, C, $\epsilon$, MinPts)
        end if
    end for
end procedure

procedure expandCluster(P, NeighborsPts, C, $\epsilon$, MinPts)
    add P to cluster C
    for each point P' in NeighborPts do
        if P' is not visited then
            mark P' as visited
            NeighborPts' = regionQuery(P', $\epsilon$)
            if sizeof(NeighborPts') $\geq \epsilon$ then
                NeighborsPts = NeighborsPts joined with NeighborsPts'
            end if
        end if
        if P' is not yet member of any cluster then
            add P' to cluster C
        end if
    end for
end procedure

procedure regionQuery(P, $\epsilon$)
    returnt all points within P's neighbor(including P)
end procedure
```

C_1 and C_2, very close together and there is a border point p that belongs to both clusters, in this case, p is assigned to the cluster that is created first. The algorithm is independent of the order of analysis of the points in all other situations.

Given the previous observation, the definition used for minPoints is no longer correct. In some cases, DBSCAN can produce clusters with fewer points than *minPoints* [44], up to the extreme case in which a cluster is formed by only one point. All other minPoints-1 points that should form the cluster could be border points in common with other clusters that have been previously analyzed. There is therefore no guarantee that each cluster is made up of at least minPoints points.

Algorithm 1, in the worst case, has a complexity of $\mathcal{O}(n^2)$ as it visits each point of the dataset at most `n` times. Despite this, in reality, the cost is determined by the number of calls to the `regionQuery` function, which is called once for each point. If we use data structures to speed up a point neighbor's computation, and if ϵ has been selected so that on average each point has $\log(n)$ points within its neighborhood, then the cost of the function is $\log(n)$, taking the algorithm's complexity to $\mathcal{O}(n \log(n))$. Saving the different distances measured between the points will speed up the time taken. However, this information occupies $\mathcal{O}(n^2)$ memory while in the standard implementation the cost in space is $\mathcal{O}(n)$. Several kinds of research have been conducted to lower the complexity of DBSCAN. In [45] some results are reported. In particular, "A Linear DBSCAN Algorithm Based On LSH" [46] and "A Fast Density-Based Clustering Algorithm for Large Databases" [47] have been proposed. They are two strategies that reduce the computation time to $\mathcal{O}(n)$ proposing anyway competitive results such as those of DBSCAN.

Parameter Estimation

A very common problem for machine learning algorithms is the choice of parameters. In the case of DBSCAN, it is possible to identify their value by following the heuristic proposed by Ester *et al.* in [43]. This strategy identifies the ϵ and minPoints values capable of determining the "thinnest" cluster in the dataset.

Let D the dataset and d be the distance between p and its nearest *k-th* neighbor, then the d-neighborhood contains at least $k + 1$ points. In most cases it contains exactly $k + 1$, it can contain more samples only if there are at least two points whose distance is the same and it less than d. We define a function called *k-dist* that maps each point in the dataset to the distance from its *k-th* nearest neighbor (*k-dist*: $D \rightarrow R$). Details on the density distribution of the dataset can be obtained by plotting the *k-dist* values in descending order, as we can see in figure 3.9 [3] where there is *k-dist* graph with multiple k values. The graph obtained is called *sorted k-dist*. If we choose the i-th point, we set the value of minPoints equal to k and we take the ϵ value indicated in the graph, we know that in the results of the algorithm all points with an equal or smaller *k-dist* value will be core points. We define a threshold point as the first point in the first "valley" of the sorted *k-dist* graph, using the ϵ value highlight by the threshold point we can define, with the maximal *k-dist* value, the "thinnest" cluster of D.

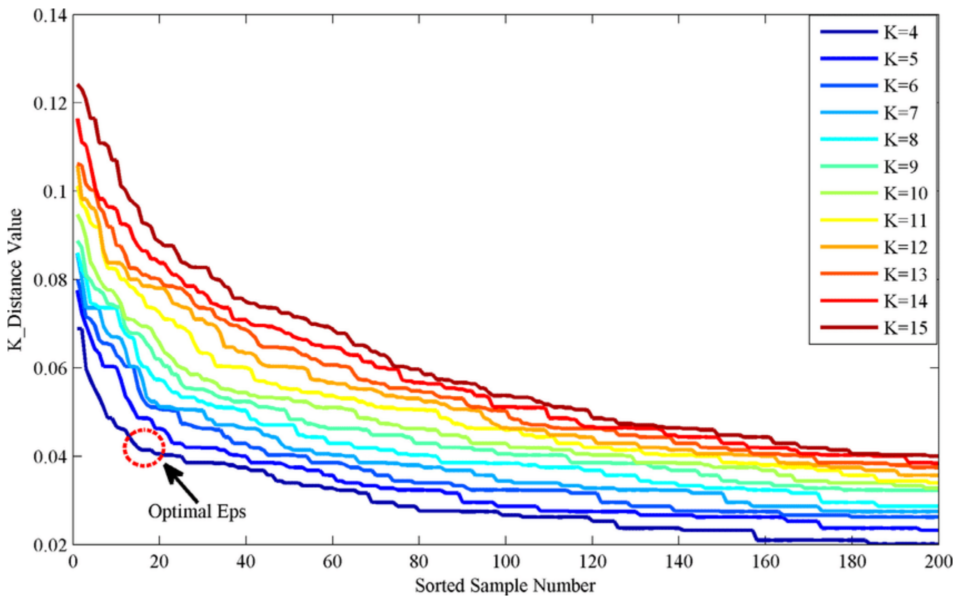


Figure 3.9: Sorted *k-dist* graph with multiple k values example

Implementing a mechanism that automatically determines the choice for these parameters can be a very stimulating challenge. In [49] A. Karami and R. Johansson propose a hybrid approach that combines DBSCAN with Binary Differential Evolution to easily and automatically determine suitable parameter values for ϵ and minPoints. According to the authors, their approach leads to precise results in determining the value of the input parameters. Another very interesting approach is the one developed in [50] by W. Lai *et al.* The authors use a Multi-verse optimizer (MVO) algorithm, which is a special variable updating method with excellent optimization performance, for optimizing the parameters of DBSCAN. Through MVO they can quickly determine the values of the input parameters that create the most accurate clusters.

[3]Image taken by [48].

Advantages and Disadvantages

DBSCAN is one of the most used clustering algorithms, the fact that it is not required to know the number of clusters a priori definitely stands out among its greatest strengths. This is certainly an outstanding function in all those instances in which the algorithm is used to gain data information. The method can classify both non-spherical clusters and outliers, based on the principle of density and not on that of distance. In this way, it is robust to noise. Another benefit is that a domain expert can easily choose its parameters or automatically determine one of the many approaches developed by [51, 52].

However, also DBSCAN has certain flaws. The key one has already been underlined and is connected to the fact that the algorithm is not deterministic. So the outcome can depend on the order in which the data is evaluated in some circumstances. Furthermore, the algorithm may not work properly in cases where there are clusters within the dataset formed with very different densities, it could consider clusters with a lower density as outliers. To avoid this, the algorithm can be iterated again, using a lower density on those points that were previously considered outliers, to verify that they are not clusters.

The quality of the formed clusters depends not only on ϵ and minPoints but also on the choice of the distance used, the Euclidean distance being definitely the most common. This may not be very successful due to the "Curse of Dimensionality", in situations where the dimensionality of the data is relatively high, but this effect is present in all algorithms that use the notion of distance.

Alternative Density-Base Methods

As previously anticipated, there are other algorithms based on the concept of density. Among these we have:

- Ordering Points To Identify the Clustering Structure (OPTICS) is an algorithm proposed by Mihael Ankerst textit et al. in [39] which is able to correctly determine clusters with different densities in a single passage. OPTICS has the same input parameters as DBSCAN: ϵ and `minPoints`, but unlike the latter it allows us to simultaneously identify a group of clusters ordered according to their density, the densest clusters will be finished first. To do this it needs a point sorting step, in which the spatially closest points become close in order. The output of OPTICS is not the clusters found but rather a different representation of the database from which it is possible to determine these clusters. The information contained in this representation allows us to identify clusters for multiple values of ϵ.

- Hierarchical DBSCAN [40] first builds a hierarchy to understand which clusters end up merging together and in what order, then for each cluster it understands whether it is better to divide it into its subclusters or keep it together. To make this decision, verify that the number of points that would belong to the two subclusters is greater than the number of points that would become outliers. These points are those that belong to the generic cluster but that would not belong to the new subclusters. If so, the algorithm produces the two sub-clusters. In this way, HDBSCAN is also able to highlight the presence of clusters with different densities within the dataset.

- A new non-parametric clustering approach is introduced by Averbuch-Elor *et al.* in [41]. Their strategy is built on the concept that each latent cluster is constituted by levels that surround its core. Clusters are indirectly separated by the external layers or border points. Contrary to the previous strategies, such as DBSCAN, where the centers of the clusters are specifically described by their densities, here a gradual peeling of the border points exposes the core ones. In the article, the authors show how the peeling process can correctly separate adjacent clusters by adapting to local densities.

Chapter 4

Related Works

In recent years there has been an incredible increase in the production and study of GPS data, highlighting the need for efficient analysis of the same in different application domains. One of these is the definition of driving behavior. To determine this, two different strategies were adopted: supervised and unsupervised learning. In the first, trained models that are able to classify new behaviors starting from a set of data with labels while in the second data do not have a label, so it is divided into groups whose elements are similar to each other and different from the elements of other groups.

Some of the works achieve, using the supervised learning approach, are that proposed by S. Chen *et al.* [53] and that proposed by H. Eren *et al.* [54]. In the first, the authors, exploiting only the data concerning the state of the vehicle, such as engine RPM, are able to determine whether the behavior is safe or not with very high accuracy. The data used in this case is very similar to the CAN data shown in section 5.2.2. The approach proposed in [54], contrary to the method reported above, is based only on data collected through mobile phones, in this way, data on the state of the vehicle are not directly available but must be obtained from the sensors present in the phones, such as the accelerometer. The authors propose an interesting work that aims to estimate the driving profile of drivers. It is able to detect risky driving patterns that can be generated by drowsiness, carelessness or traffic violations. To do this, a simple Bayesian classifier was used. Despite the excellent results obtained, the authors envisage further experiments using also CAN and environmental data. Another noteworthy approach is the one proposed by G. Li *et al.* in [55]. The authors construct driving operational pictures (DOPs) and base their analysis on these. These images are nothing more than the collection for 60 seconds of the values of 42 different features, among these, there is both information regarding the driver and information on the state of the vehicle. Each of these images is manually assigned a class. Convolutional neural networks

(CNN), long short-term memory (LSTM) network, and pretrain-LSTM are used to classify driving behavior. The studies proposed in [55] show that CNN is the best model. The strategy reported by P. Brombacher *et al.* in [56], although based on supervised techniques, is very similar to that proposed in the entire pipeline. The authors, starting from data collected by devices based on a Rasberry Pi, classify the maneuvers performed by drivers using a neural network. Then, based on these maneuvers, they assign a KPI on a trip to a driver, this score is an index of the driving quality. Through these indicators, they are also able to automatically assign an average score to the various maneuvers. This way they can determine what attitudes lead to a particular KPI. This procedure is very reminiscent of the one developed in this thesis, with the only difference that the work proposed by us, besides obviously being based on unsupervised techniques, also takes into account the environment in which the trip was carried out.

Since defining correct driving behavior is very complicated, building datasets with labels large enough to allow model training is extremely expensive. For this reason, there are few works in the literature that address the problem of defining driving behavior with a supervised approach.

Since the data provided by K-Master does not have the labels, in this thesis more attention has been given to works that exploit an unsupervised approach. In the literature, the topic of how to model GPS data has been attached several times. A standard approach to working with this type of data is to identify and organize in groups the trajectories traveled, where trajectories mean a set of temporally ordered positions. A noteworthy work in this context is [57] in which the authors observe how the existing clustering algorithms simply group objects whose trajectories are completely common without considering that two vehicles may have traveled part of their trip together. For this reason, they propose a framework capable of partitioning a trajectory into sub-trajectories and then analyze these sub-parts. Another interesting approach is the one developed by A. Tietbohl *et al.* in [58]. The authors go beyond the classic geometric approach to problems related to trajectories, linking semantic information to the points covered. So they assign different importance to the various points of the trajectory, while in the other works all points have the same relevance. These two works model data in a very similar way to ours.

However, the work done goes beyond the simple grouping of trajectories, trying instead to define a driver behavior for each trip based on similar ones through clustering operations. Driving behavior is a complex concept, it describes how the driver handles the vehicle in relation to the surrounding environment. For this reason, not only GPS data were used, but also data relating to the state of the vehicle and the environment. A more detailed description of the data used can be found in chapter 5.

Examples of researches where an unsupervised approach was used to determine driver behavior are [59, 60, 61]. In the first, the authors implement an alternative version of the DBSCAN algorithm called Iterative-DBSCAN that allows them to highlight negative behavior on a large dataset of real data. The proposed algorithm can be summarized in three steps: (1) modeling the data; (2) splitting the data into subsets; (3) running I-DBSCAN on each group. [59] proposes a pipeline very similar to ours, but their work focuses more on the application of I-DBSCAN. In the second K.T Chen and H.Y.W. Chen explored the possibility of identifying driving styles using also driving parameters such as age, gender, driving experience. They use Partitioning Around Medoids to build the clusters and then apply the PCA to understand the structures of the clusters. In this way, they define three different driving styles. The results showed that all three attributes examined had an impact on how the trips were clustered. Thus suggesting that driving style is also influenced by characteristics that describe the driver. However, this type of information, not being available among the K-Master data, was not taken into consideration in the performed work. B. Higgs and M. Abbas in [61] propose instead a two-step algorithm for segmentation and clustering. In the first step, the trips are divided into blocks of equal duration while in the second they are clustered using the k-mean algorithm. A similar approach was developed in this work, but unlike them, we divided the journeys based on the point of departure and arrival and we used the DBSCAN algorithm as k-mean does not allow to identify the outliers among the data.

Contrary to the unsupervised works presented so far, where data is collected from devices in vehicles, in [62] the authors used a large set of data collected from users' smartphones. V. Sokolov *et al.* Demonstrate how to use unsupervised learning to identify distinct driving patterns and how to group drivers based on this.

Chapter 5

Case Study

This work was performed out by analyzing the drivers' behavior of a part of the fleet managed by K-Master Srl, which is a company subject to coordination and control of Telepass SpA. In particular, the data collected by 30553 vehicles in June 2020 was analyzed. The data used for this work can be grouped into three macro-categories: the type of vehicles, vehicle and contextual data. The first two types were provided by K-Master while the last one was collected using APIs.

In figure 5.1 some of the data delivered by K-Master are shown. Each dot represents a GPS point, the color indicates which vehicle it belongs to. As we can see from the map most of the fleet moves to Italy and central Europe but there are also many cases in southern England. The map also shows the presence of GPS data in the sea, this can be due both to the presence of errors and to the fact that the vehicle was on a ferry. In both cases, these values will have to be eliminated to conduct a correct analysis.

5.1 Vehicle Type

Since the vehicles' behavior varies according to their type, K-Master has provided us with a partial registry of their fleet. This document contains the following information for each fleet's member: manufacturer, model, year of construction, type of vehicle and power source. The only useful feature for the analysis is the type of vehicle.

5.2 Vehicle Data

In each vehicle of the fleet is mounted a device that collects different kinds of information: a first basic set common to any device, later called GPS data, and a

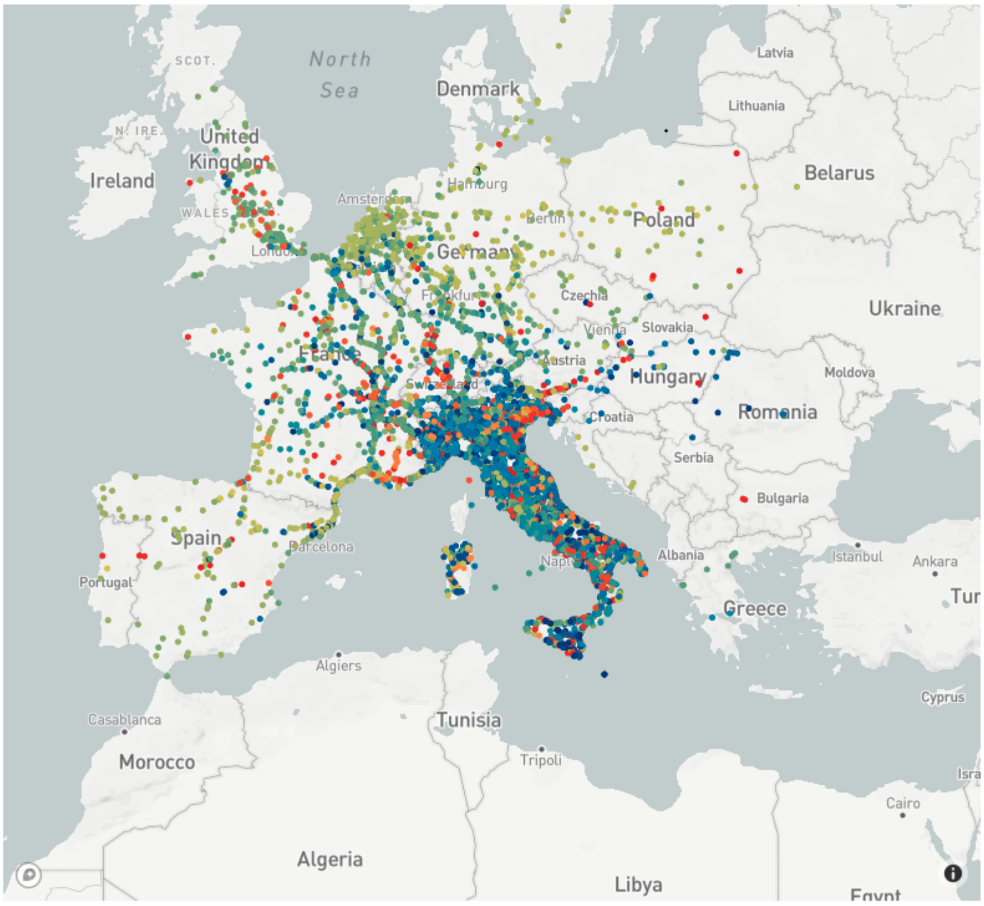


Figure 5.1: A subset of the GPS data provided by K-Master.

second one, later called CAN data. If the device can send CAN data these are sent with GPS ones in a single message otherwise this contains only GPS data. Within a message containing CAN data, there can be different groups of information, as the sampling frequency depends on the type of data we are collecting. Since, based on the mounted device, a different service is offered for the vehicles, these are divided into two categories: the ones that use devices that send CAN data, called *Premium*, and the ones that only send GPS information, called *Business*.

The data is only acquired when vehicles are on with a variable frequency that changes accord to the device that collects them. Furthermore, the mounted devices send signals as soon as the engine is switched on, even before the sensors can carry out the necessary revelations. For this reason, and because sometimes the sensors collect a wrong value, there are many dirty values inside the data. Moreover, some vehicles may not send data because there is no signal on the stretch of the road traveled. This information is not lost but is sent as soon as possible, so there are also times when information is received with a high frequency.

5.2.1 GPS Data

The first set of data is common to all vehicles and contains the basic information needed to locate them. We have the position, given by a couple of values (*latitude*, *longitude*), the time when the data was produced and the one when they were invited since, as we said before, these values can be different. Basic information also contains the actual speed and the total distance covered.

Using this information it is possible to define and display on a map which trips are made by each driver. In figure 5.2 there is a graphic representation of a journey.

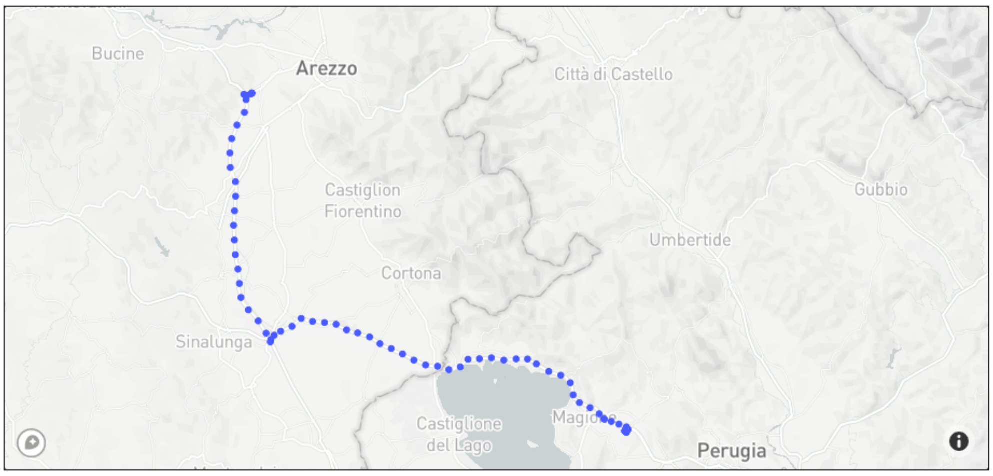


Figure 5.2: Example of trip visualization.

5.2.2 CAN Data

CAN data contain technical information such as the status of the various mechanical components. They are acquired only by those vehicles that use the *Connected* service. This information is collected by different devices and for this reason, the vehicles that send them can send both sets of different data and the same data but with different encodings. The sampling frequency of these measurements depends on the information type sent. The type of device used and the sampling frequency determine the data contained in a packet sent. As we said above, the messages transmitted when the vehicle is started are generally wrong as they are sent before the sensors can perform the measurements. In these cases, either old values or random ones are forwarded. For these reasons, before we can proceed with their use, an analysis is necessary to verify the frequency of correct data.

5.3 Contextual Data

Not only indicators coming from the vehicle but also the ones that describe the environment define the driving behavior. Indeed it should change if it is a runny or a sunny day. For this reason, weather information has been added to the data provided by K-Master. Furthermore, since it was not present in the GPS data, information regarding the difference in height in the various trips were added.

5.3.1 Weather Information

The speed at which we make the trip or the efficiency of braking are highly influenced by the weather conditions with which the journey was conducted. The same performance of two trips on the same route with completely different weather conditions cannot be considered similar to each other. For this reason, within the pipelines, a tool has been implemented to acquire weather information based on Dark Sky API [63]. The service offered by Dark Sky is global but, due to the availability of data, some features are only supported regionally. In any case, the regions in which the K-Master fleet operates are covered by this service. Using this API it is possible to trace the meteorological information relating to a certain position for a given instant of time. In case some information is not available for that moment, a more generic version is returned for the entire hour. If this is also not available, an even more generic version is returned for the whole day.

In general, defining an optimal situation, in terms of weather conditions, for driving is not trivial. We might think that the best is the one with a clear sky and during the day, but in reality, if we are driving towards the sun, visibility is compromised. Another aspect that influences the correct driving style to be adopted is the quantity and type of precipitation. Mainly for these reasons the following thirteen features have been chosen

- `cloudCover`, a continuous value between 0 and 1, indicates the percentage of the sky obscured by clouds.
- `icon`, a discrete value between 0 and 3 that describes the visibility based on the current weather. It is 0 in case of clear day/night or wind, 1 in case of cloudy day/evening, 2 in case of rain and sleet, and 3 in case of snow and fog.
- `precipAccumulation`, inches of snow expected. In case of snow, drivers should be more cautious.

- `precipIntensity`, the intensity of precipitation expressed in millimeters per hour. In the event of heavy rain, phenomena such as water planning can make it difficult to stop the vehicle and visibility may also be compromised. For these reasons a lower speed is expected if this feature and the next one assume high values.
- `precipProbability`, the likelihood of precipitation occurring.
- `precipType`, discrete value between 0 and 3 indicating the type of precipitation. It is 0 if there is no precipitation, 1 in case of rain, 2 in case of sleet, and 3 in case of snow. The increase of these features compromises the driver's visibility.
- `sunriseTime`, sunrise time of the current day. This feature and the next one indicate whether the journey was made during the day or night. It is common knowledge that we should be more cautious at night.
- `sunsetTime`, sunset time of the current day.
- `temperature`, the air temperature, expressed in Celsius degrees. If temperatures fall below zero, ice sheets may be present along the road, in these cases, it is necessary to drive more carefully.
- `visibility`, the average visibility expressed in kilometers, for a maximum of 16 km. It is an indication of how far the driver can see on straight roads.
- `nearestStation`, the distance from the nearest weather station that helped provide the information. Indicates how reliable the acquired data are. The closer the station is, the more likely it is that the data are true.
- `numberHourlyFeatures`, the number of previous features acquired with an accuracy of the hour and not of the instant. These features and the next one indicate how accurate the acquired data is in terms of time. The shorter the time interval over which the forecast was made, the greater the probability that the predicted meteorological condition occurred.
- `numberDailyFeatures`, the number of previous features acquired with an accuracy at the level of the day and not of the instant.

5.3.2 Altitude

The pipeline has a tool based on Jawg API [64] to acquire altitude information. Using it was possible to find the altitude relative to a certain position. It is expressed in meters above sea level and is not added to the various records as it is found, but the variation concerning the previous record is indicated. The calculation of the altitude deltas is relative to the order of the records, so it will be positive in case of uphill and negative in case of downhill.

Chapter 6

Proposed Approach

This chapter explains in detail the solution proposed, before proceeding it is necessary to introduce some definitions:

- given a vehicle, a **trip** is the set of consecutive data whose temporal distance is not greater than a certain threshold (`time delimiter`). A new trip is formed whenever the time interval between one data and another is greater than `time delimiter`.

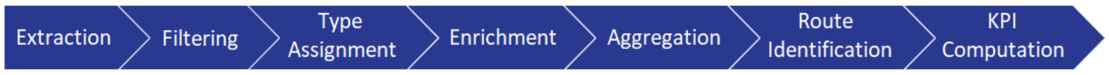
- a trip is **Connected** if it is associated with a vehicle with a *Connected* service transmitting a sufficient number of CAN Bus messages. All the non-rich trips are classified as **Business**. Our approach defines trips as *Business* or *Connected* and not vehicles as some vehicles may, over time, switch from one service to another.

- we model the globe using a geographical grid of squared **cell**s.

- a **route** is defined as the set of trips starting from a given cell and ending in another given cell.

The work aims to implement a highly user-configurable framework that divides the various driving behaviors present within a route and assigns a Key Performance Indicator (KPI) to each journey that makes up this route. The developed framework is composed of seven steps. The thesis focuses on the first six and the first part of the last one, placing particular attention on the data preprocessing phase and on the clustering one. For each step one or more Python scripts were implemented. All the steps, except the first one, have a configuration file containing a series of global variables that can be modified by the user.

The implemented steps compose a pipeline. The output of one step is the input of the next one. Some parts of the pipeline are optional, the subsequent phases



can be performed both if these parts have been performed and if they are not. The various steps can be summarized as follows:

1. Raw data extraction from the database.

2. Raw data filtering. The data is grouped in a trip and enhanced with other information derivable directly from the data itself. Also, the grid model is applied, so that each record falls into a unique cell of the grid. Next, data is filtered based on configurable specifications, at both single-record level, to remove anomalous rows, and trip level, to consider only the relevant trips during the analysis.

3. (Optional) Vehicle type assignment. Each vehicle is assigned its type. To do that, we use one of the strategies explained in section 6.3. This step is optional, if it is performed, in the following ones the analysis can narrow on only one of the two types.

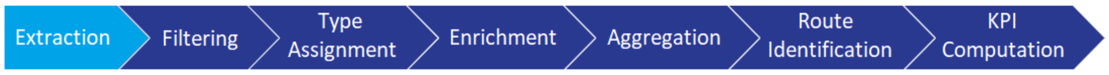
4. (Optional) Feature enrichment. To further enrich the data position and the timestamp related to each record are exploited, adding information such as weather conditions and altitude. This part of the pipeline is optional, subsequent steps can be run either the contextual features are present or not.

5. Trips aggregation. The records belonging to the same trip are grouped into a single row that represents the trip itself. The indicators that describe each GPS point are aggregated to produce new ones relating to the entire trip.

6. Most covered routes identification. The trips are grouped into routes, based on the departure and arrival area. Trips that belong to the best route are copied into specific folders to facilitate the following analysis. It is also possible to print the raw data of the trips of the route under analysis on an interactive map.

7. KPI computation. On the trips of a route, the DBSCAN algorithm is applied, the trips that are not considered outliers are divided into train and test set. A model is trained on the first group and is used to determine the KPI of the elements of the second group.

The following sections give more details on the steps listed above.



6.1 Data Extraction

The first step of the pipeline is used to parse the original raw data and transform it into a format suitable for analysis.

The current KMaster data is stored in MYSQL server databases. Our pipeline processes data from three relational tables:

- **sat_msgPos.** This table stores GPS messages. The *idMsgVpr* attribute is a *foreign key* for the CAN header table.
- **sat_vpr.** The CAN header table. Apart from timestamp attributes, this table is used as a bridge table between the GPS messages and the CAN Bus messages. The *idMsg* attribute is a *foreign key* for the CAN Bus message table.
- **sat_vprItem.** The CAN message table. Each record is associated with a CAN header (from table vpr) and contains the information of a specific CAN message.

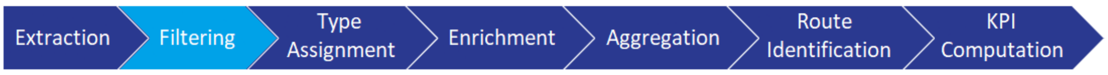
The total weight for a month-worth data is over 70 GB. Due to the large amount, the first step is designed to split the initial raw messages by vehicle ID. As such, the phase produces one CSV file per vehicle ID (i.e. *idVeh* column in the relational database). Furthermore, to reduce the workload in the next step, it is possible, through a flag, to discard all the features that will not be used.

The benefit of adopting this format is two-folded. On the one hand, having separate source files enables the parallelization of the pipeline execution. In a cluster-based scenario, vehicles can be batched and assigned to different nodes (even though it would require an alignment phase later on the pipeline). On the other hand, users can carry out selective experiments. Among the possibilities, multiple models can be trained only on a smaller portion of the fleet, or a set of vehicles can be excluded/included from the pipeline without additional effort.

The module does not interface directly with the MYSQL server, as network bottlenecks could arise. Hence, it processes the textual file representations of MYSQL tables, as they are produced by the *mysqldump* utility.
This step, together with the following one on filtering, is the most computationally expensive. It is mainly due to the use of a single multiprocessing node to load

data into memory and split it a few batches of vehicles at a time. However, per its design, this stage is likely to benefit the adoption of established frameworks for Big Data such as Hadoop or Spark, where the data is loaded in different nodes and the process is parallelized.

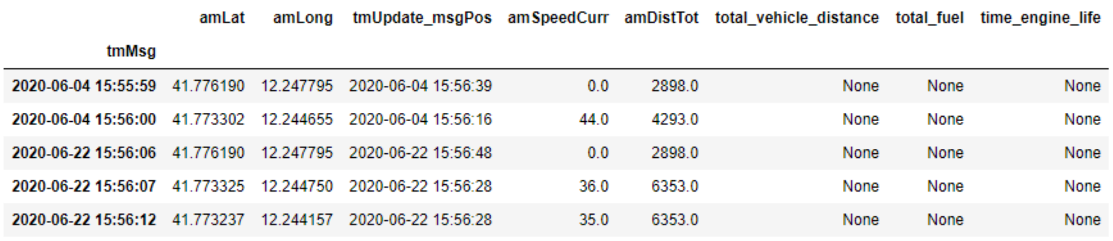
6.2 Raw Data Filtering



The second step takes in input the files containing the data separated for each vehicle. For each of them, it generates in output a corresponding file containing only the records that passed the filtering, all with some added information derivable directly from data itself. The user can choose whether to perform the study on all files present or a subset. In the latter case, he must list the ids of the vehicles on which to conduct the analysis.

6.2.1 Feature Engineering

The first step is to redefine the columns of the files that characterize a vehicle. We begin grouping the records by their ID message. From each group, we get a single row with only the GPS columns concerning timestamps, coordinates, speed and distance and we use CAN information to have a column for each CAN feature of interest. Moreover, the values inside these CAN columns are not the original ones but they are the delta to the previous row. The first row gets a void value. Before being inserted, these deltas are checked by verifying that they fall within a limit (given that the order of magnitude of the variations is known and is around the unit). In this way, we do not consider the anomalous values present in some CAN lines of many vehicles. The result after this phase is represented in figure 6.1.



	amLat	amLong	tmUpdate_msgPos	amSpeedCurr	amDistTot	total_vehicle_distance	total_fuel	time_engine_life
tmMsg								
2020-06-04 15:55:59	41.776190	12.247795	2020-06-04 15:56:39	0.0	2898.0	None	None	None
2020-06-04 15:56:00	41.773302	12.244655	2020-06-04 15:56:16	44.0	4293.0	None	None	None
2020-06-22 15:56:06	41.776190	12.247795	2020-06-22 15:56:48	0.0	2898.0	None	None	None
2020-06-22 15:56:07	41.773325	12.244750	2020-06-22 15:56:28	36.0	6353.0	None	None	None
2020-06-22 15:56:12	41.773237	12.244157	2020-06-22 15:56:28	35.0	6353.0	None	None	None

Figure 6.1: Result After Column Selection and Transformation.

Next, for each of these records, the delta of space traveled and time elapsed to the previous row are calculated, just like the CAN features. In particular, it is assumed that the distance between two consecutive records is traveled as the crow flies. Therefore, unless there are errors in the data, it is less than or equal to the distance actually traveled.

Furthermore, each of them is assigned the trip identification to which they belong based on the time delta just calculated. If this is less than a certain threshold (`time delimiter`) then the record in question belongs to the same trip as its predecessor. Otherwise, this record is the first of a new trip. Finally, to be able to assign to each record the ID of the cell in which it is located. To do this a grid made up of square-shaped cells, each with a unique ID, is determined on the world map. To each record is assigned the cell ID that contains its position and the indices that identify this cell within the grid. Cell size is a configurable parameter.

6.2.2 Record Filtering

Subsequently, the following filters are applied to each record, the user can choose for each of them whether to activate it or not.

- Filter for outliers, if the coordinates have unexpected values the record is deleted. By outliers we inted either a coordinate that deviates too much from the mean between all records of the same vehicle $\pm$ the standard deviation multiplied by a certain user-specified coefficient or a pair of coordinates both 0°. We define it to avoid being considered records in which the coordinates have been reset due to GPS or data transmission problems.

- Filter on the maximum speed, if the speed necessary to cover the delta of space in the delta of elapsed time obtained is too high, the record is removed. The maximum speed is chosen by the user, it is advisable to set it up high enough to detect certain errors (standard value 200 km/h).

- Filter on positions at sea, if a pair of coordinates results in the middle of the sea, the corresponding record is removed so as not to consider any errors or journeys of vehicles that are embarked. This information is found using the OnWater API [65].

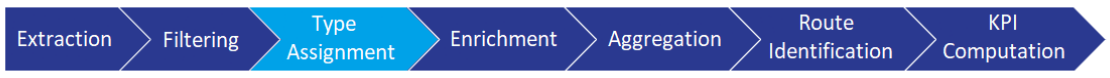
6.2.3 Trip Filtering

Lastly, filters are applied at the trip level, to remove all those records that belong to a trip that did not pass the selection. This screening is carried out to avoid considering both trips of too short duration and distance and trips that are not detailed enough due to the scarce amount of records that represent them. The filters applied are:

- Filter based on duration, trips with a duration less than a certain threshold are discarded. It allows you to discard journeys relating to insignificant movements.
- Filter based on traveled distance, if the sum of the distances as the crow flies between the various positions characterizing a trip is less than a certain threshold the trip is discarded. It allows you to discard trips related to short maneuvers, as can happen in a parking lot or an establishment.
- Filter by average speed, trips with an average speed below a certain threshold are discarded. It allows you to discard trips where the vehicle has actually been running for a long time without being moved.
- Filter on the minimum number of lines, if the number of trip rows is less than a certain threshold, then these records are all removed. It avoids considering trips that are too short or long with a too low frequency of information transfer.

All the thresholds of the filters just described can be set by the user.

6.3 Vehicle Type Assignment



Since the vehicle's behavior is influenced by its type, it could be useful to identify it before carrying out the analysis. The third phase implements this idea. It takes as input the files generated by the previous step and divides them according to the type of vehicle that they represent. To do this we exploit the information contained in the registry. As in this file, there are various vehicle categories, these have been grouped into two macro-categories which are:

- HEAVY vehicles, that is those whose `TYPE` field in the registry file is one of: TRUCK, TRAILER, BILICO, Eurocargo, e Stralis.
- LIGHT vehicles, that is those whose `TYPE` field in the registry file is one of: AUTOVETTURA, 35C12, 35C15, 60C15, Maxcity.

6.3.1 Strategies for Determining the Type of Vehicle

From these values, we can see how the value of the TYPE field is often absent in the current registry. For this reason, it was necessary to implement a mechanism capable of assigning the type to each vehicle. Two strategies have been proposed: the first based on heuristics and the second based on machine learning algorithms (Random Forest and Gradient Boost).

Heuristic Strategy

In choosing the type to assign to each vehicle without this information, due to the lack of completeness of the data, two heuristics based on the average and maximum speed were tested. For each of the tested speed thresholds, if a vehicle is found to be above it then it will be considered LIGHT, otherwise, it will be categorized as HEAVY.

Machine Learning Strategy

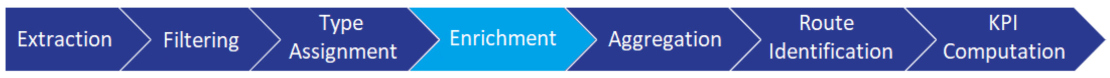
An alternative to using heuristics is to train models to predict the vehicle's type. The training takes place using as train set the vehicles ' data whose type is known from the registry.

The usage of GPS data as input for the models is not particularly useful. For this reason, we exploit the data of vehicles whose macro-type can be deduced from the registry. These are the vehicles whose TYPE field takes on one of the values previously listed. For each of them, the features useful to distinguishing the type at the trip level are calculated. Contrary to the heuristic strategy, in this way it is possible to consider, in addition to the average and maximum speeds, also the duration of the trips, the distance traveled, the number of breaks present within the trip and the time slot in which the trip has been made. After that a further aggregation is performed at the vehicle level, determining the maximum, minimum and average value for each of the previously calculated features.

The models used are Random Forest (RF) and Gradient Boosting (GB) classifiers. We have chosen to use these models as they both increase the accuracy of decision trees but with opposite strategies. We wanted to find out which of the two performed better. Leave One Out CV (LOOCV) was used to obtain more robust results. LOOCV consists of dividing the train set into k-folds where k represents the number of elements it contains. Thus, the classification algorithm is performed once for each fold, using all other folders as a training set and using the selected fold as a test set. The model results are the average of the various iterations. In this way, the results obtained do not depend on the split of the data in the train and test set. Accuracy, precision and recall for both classes were calculated to evaluate the two classifiers and understand which of the two performs better.

Regardless of the strategy used, the third step searches for each vehicle for its type in the register or in a file containing vehicles whose type has already been predicted. For vehicles for which this is not available, calculate the same vehicle characteristics calculated for the vehicles contained in the register, and apply the chosen strategy to determine the type.

6.4 Data Enrichment



The fourth step takes as input the files generated by the previous phase and as output generates an extended version of them with all the new features relating to weather and altitude. This step is optional, meaning that the rest of the pipeline can be executed on both the output of this step and the output produced by the phase in the previous section. If the user has differentiated the data by type of vehicle, through a flag, he can choose whether to enrich one type or both.

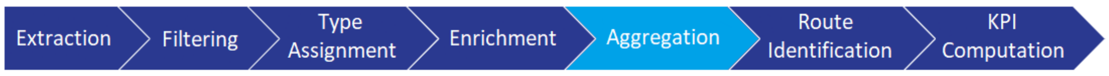
6.4.1 Enrichment with Contextual Features

In the first part of the step, the records are enriched by adding the meteorological features listed in section 5.3.1. Making an API request for each record is very time-consuming, so to overcome this problem a mechanism has been implemented. Its base idea is: given a certain record without meteorological information it verifies if there is another point very close both in space and in time already in possession of this information. If present the weather data are retrieved from that record, otherwise a new query is submitted. To define if two points are close in space and time the user can define two thresholds.

This system speeds up the process and saves a lot of API access. The data retrieved with this solution can only be inherited from a record that has requested it directly from the API, to avoid forming a too long chain of information passing from one record to another.

In the last part of the step, the data is enriched by adding the difference in height between two records. Due to the nature of the information acquired, it is not possible to use the mechanism of the previous point, making this part of the phase relatively slow.

6.5 Trip Aggregation



The fifth step takes in input a set of files, one per vehicle, and collapsing all the rows belonging to the same trip in one unique one. It generates a unique file containing all the trips of all vehicles.

6.5.1 Aggregated Features

The information contained in the output row is described below, grouped in three main categories.

Meta Information

These are boundary information, such as the various IDs and timestamps, useful for a better representation but not for calculating both clusters and KPI. For this reason they are not taken into consideration by the machine learning algorithms.

- `idVeh` and `tripID`, are respectively the vehicle and trip identifiers. Together they can uniquely identify a trip if there is only one file for each vehicle.
- `firstCellID` and `lastCellID`, are the identifiers of the starting cell and the arrival cell.
- `firstTimestamp` and `lastTimestamp`, are the timestamps of the first and last record.
- `readingsCount`, represents the number of records that characterize the trip.
- `firstLat`, `firstLon`, `lastlat` and `lastLon` are the coordinates of the start and end points of a trip. They are useful in the next phase for illustrative purposes only.
- `avgNearestStation`, `stdNearestStation`, `maxNearestStation` and `minNearestStation`, they are derived from the `nearestStation` feature and correspond to mean value, standard deviation, minimum and maximum value. They can be used to validate the rest of the weather features but not for the calculation of the KPI.

Main Features

They contain the basic information that characterizes the trip under various aspects. These are present in all analyzes.

- `vehType`, represents the vehicle type that carries on the trip. If phase described in section 6.4 was not performed this feature has `None` value.
- `membership`, represent the trip type. It can be *Connected* or *Business*, its value depends on the number of occurrences of the CAN features.
- `elapsedTime`, represents the duration of the trip expressed in seconds, obtained using the timestamps of the first and last records of the trip.
- `microStop5`, `microStop10` and `microStop15`, represent the number of pauses done by the maximum duration, in minutes, of the number they have in the name and the minimum duration of the number of the previous feature (for `microStop5` the minimum pause duration 0). These pauses are identifiable thanks to the delta of elapsed time calculated by the first phase.
- `totDist`, represents the total distance traveled, obtained starting from the raw feature `amDistTot`, subtracting the value of the first record of the trip from that of the last.
- `totStraightDist`, represents the total distance traveled as the crow flies, obtained by calculating how far the coordinates of the first record of the trip are from those of the last.
- `avgSpeed`, `stdSpeed`, `maxSpeed` and `minSpeed`, derive from the `am-SpeedCurr` feature present in the raw data and correspond to mean value, standard deviation, minimum and maximum.
- `percentageHoliday`, indicates the percentage of travel time spent on a public holiday (Sunday or national holiday).
- `percentageDay`, indicates the percentage of travel time spent during the day. The day is considered as the time between sunrise (feature `sunrise-Time`) and sunset (feature `sunsetTime`).
- `avgSlotDay`, `avgSlotRus` and `avgSlotNig`, represent the percentage of travel time spent during the daytime, rush hour and night time slots respectively. The ranges ending these slots are user configurable.
- `totUpHill` and `totDownHill` they are the sums of the deltas of altitude respectively of uphill, the positive ones, and of downhill, the negative ones.

Contextual Features

They contain information derived from the weather. For each of the features listed in section 5.3.1 and not used in the calculation of the main features, the average value, standard deviation, minimum and maximum have been calculated. These may or may not be present in analyzes based on the value of a user-configurable parameter.

6.5.2 Values Computation

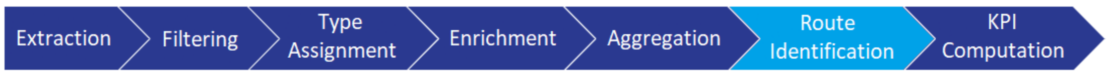
The average values obtained from many of the GPD point features have been calculated by weighing the addends that compose them based on the temporal distance between the record associated with this addend and the records relating to the previous and subsequent addend. In general, t_{i-1}, t_i, t_{i+1} are the values of the timestamp for three consecutive records, the weight of the i-th record will be:

$$w_i = \frac{(t_{i+1} - t_{i-1})}{2 \cdot tot\_time}$$

where tot_time indicates the total time of the trip, obtained with the difference between the timestamps of the last and the first trip record. As for the first and last record, their weights will be respectively:

$$w_1 = \frac{(t_2 - t_1)}{2 \cdot tot\_time} \qquad w_n = \frac{(t_n - t_{n-1})}{2 \cdot tot\_time}$$

6.6 Most Covered Route Identification



The purpose of this step is to create a filter able to show to the user the routes of greatest interest in his fleet. To do that, we realize two steps: the first assigns a score to each route and classifies them based on this, while the second interactively draws the map of the trips that make up the route.

6.6.1 Best Route Identification

The first step receives as input a file generated in the fifth step, which contains a set of trips, and groups this set according to the route to which the trips belong. Then apply the following filters, eliminating routes that do not respect them.

- Filter by vehicle type. The user can choose to delete all those sections that do not contain at least a certain percentage of HEAVY or LIGHT vehicles.
- Filter by trip type. The user can choose to delete all those routes that do not contain at least a certain percentage of *Business* or *Connected* trips.
- Filter by the number of vehicles. The user can choose to delete all those sections that do not contain at least a certain number of vehicles.

Finally, it assigns a score to each route and sorts them. The user can indicate the value with which the routes are ordered. The possible values are:

- **connected**, the route with the maximum number of *Connected* trips are selected.
- **veh**, the route traveled by the maximum number of vehicles is selected.
- **trip**, the route with the maximum number of trips is selected.

Furthermore, the user can decide to keep only those sections that start/end in a city. To understand in which city a GPS position is located, the geolocation service offered by LocationIQ [66] is used.

Since many of the vehicles within the K-Master fleet are service vehicles operating within the Fiumicino airport, the user can decide to remove those trips that start or end near the airport. It is highly recommended to remove such trips as the driver behavior inside an airport is certainly very different from those you might have on the road.

The user can analyze together trips belonging to similar sections, i.e. with the start and end cells close to each other, increasing the neighborhood of the cells. Where neighborhood is mean a positive integer number used to build a macro-cell. Given a reference cell, a macro-cell is that set of cells for which the maximum distance between the "center" cell and the others is at most "neighborhood" cells. In figure 6.2 there is a graphic representation of the neighborhood concept. In detail, given a route determined by the pair of starting cell and arrival cell, all trips that start from the macro-cell centered on the starting cell and end on the macro-cell centered on the arrival cell are considered to belong to that route. It is useful in those cases where the number of trips of a certain route is too low to conduct an analysis.

The neighborhood concept was introduced to speed up the analysis as its increase is equivalent to performing the analysis with larger cells. The change in the size of the cell affects all the steps of the pipeline, forcing the user to repeat it all, while the change in the neighborhood affects only the last two steps.

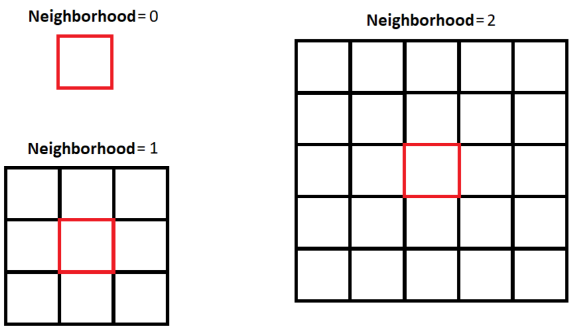


Figure 6.2: Example of macro-cells for different neighborhood values.

6.6.2 Graphical Representation of Routes

Using the second step of this phase, it is possible to draw on a map the trips that belong to the route specified by the user. If no route is specified, the one with a higher score is shown. The map created allows us to select the trips based on their type (*Business*/*Connected*) and based on the type of vehicle (HEAVY/LIGHT) that made them. We can also view the data of each GPS position by hovering the mouse over it.

6.7 KPI Computation

Extraction
Filtering
Type Assignment
Enrichment
Aggregation
Route Identification
KPI Computation

In the last step, two operations are performed, initially, the trips belonging to the same route are divided into groups, through the use of the DBSCAN algorithm, after each trip that is not an outlier is assigned a KPI. This last part is not the subject of this thesis.

6.7.1 Deploy simulation

To simulate a real situation, the trips of a route are divided into two parts: train set and test set. The split is done based on a date. All the data before are in the first group while the ones after are in the second group. The group of features called meta information in section 6.5.1 is removed from the data as it does not contain information useful for conducting subsequent analyzes. Next, all the *Business* trips are removed from the training set. The testing set is split again into *Connected* and *Business* trips to evaluate them differently. Before apply cluster algorithm data are normalized using a scaler fitted on training set feature values.

6.7.2 Clustering

In this phase, the DBSCAN clustering algorithm is used to process the train trips. As mentioned in section 3.3.3 the parameters to pass to the algorithm are two but one of them can be chosen according to the other. The step determines the best ϵ based on the value of minPoints chosen by the user. We must also set which distance to use, the default is the Minkowski one.

DBSCAN locates the outliers within the route and divides the remaining points into clusters. All points not belonging to the clusters are considered anomalous driving behavior and are discarded.

Determination of ϵ

A mechanism based on the one proposed in [43] has been implemented to determine the ϵ value as a function of the number of minPoints chosen by the user. However, contrary to this, the detection of the elbow of the curve is automatic.

Using the chosen minPoints value, for each trip belonging to the route under analysis, the ϵ distance between this and all the other trips of the train is calculated using the chosen metric. Subsequently, a histogram is constructed having the distances on the ordinate axis and the number of points on the abscissas with at least minPoints points within the indicated distance, or the number of core points that would be defined using that distance. In figure 6.3 there is an example of this graph calculated on a group of trips belonging to a route.

The ϵ distance is chosen by identifying the elbow in the graph. This elbow is automatically detected by observing the variation in value between two consecutive columns of the histogram. If this delta is greater than the average of the previous deltas multiplied by a certain coefficient greater than 1 the point is considered as the knee of the curve and the corresponding distance is selected as ϵ.

Most Important Features Identification

To underline the difference between elements of clusters and outliers, the framework automatically identifies the features that most contributed to separating the trips and shows their distribution. The latter is represented in the form of a histogram where the columns are arranged in order of value, each one represents an element and is colored differently according to the cluster to which it belongs. Among

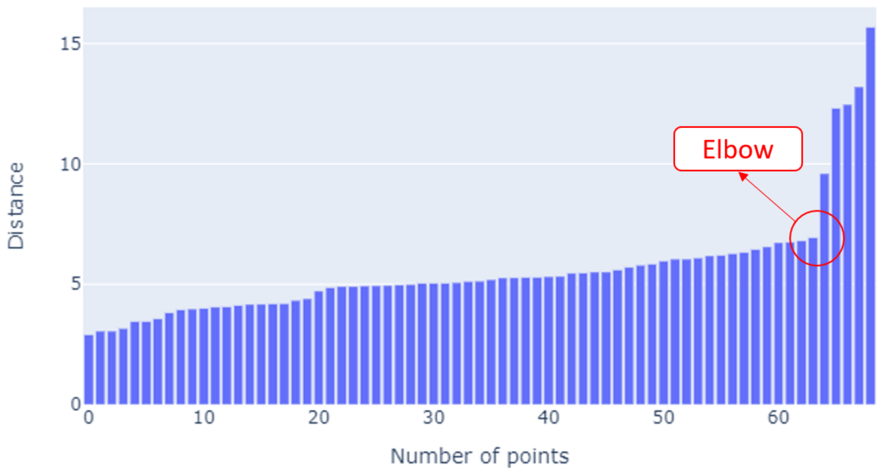


Figure 6.3: Example Graph of the Number of Core Points as a Function of Distance and MinPoint.

these, those representing the outliers are highlighted in red. In figure 6.4 there is an example of such a graph, as it is possible to see most of the outliers maintained an average speed greater than the travels of the cluster.

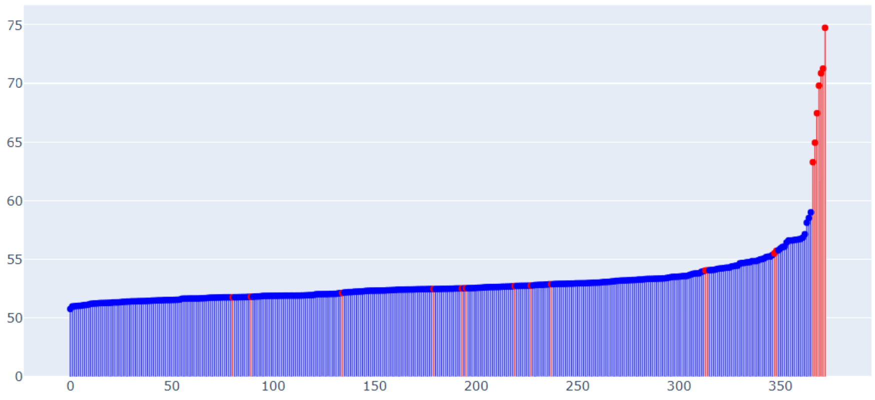


Figure 6.4: Example of Distribution of Average Speeds.

To identify the features with the most relevant distribution, a score is assigned to each of them in such a way that a higher score indicates greater relevance. We base the score calculation on the difference in value that exists between neighboring columns with different labels. Before obtaining this difference, all the values of the various distributions are scaled between zero and one in such a way that the comparison between scores of different features is not influenced by the domain scale of the latter. Each column is analyzed individually, looking at the closest K columns to both the right and left in the histogram. For each of them, if the label differs from that of the reference column, the difference in value is added to the score. Parameter K is also automatically identified, searching in all distributions for the longest sequence of consecutive elements belonging to the minority class and using this length as the number of neighboring columns to observe.

This strategy, however, is not present in the latest version of the framework, so the results present in section 7 do not show these graphs.

6.7.3 KPI through KDE

In the second part of the step, a Kernel Density Estimation (KDE) technique is applied on the train set data is evaluated. KDE is a strategy capable of assigning driving behavior to a trip by determining the probability distribution of the driving style in the neighborhood of the journey under analysis. The size of the neighborhood to consider is a parameter: as KDE expands, it will consider less and less similar behavior. The behaviors of the medoids, being the most common ones, are considered optimal driver behaviors. Subsequently, for each test trip, a KDI is calculated based on the medoids found and the KDE.

Chapter 7

Experiments

This chapter summarizes the results of the experimental campaign aimed at investigating the data provided by K-Master and the impact of system parameters on the creation of clusters. The experimental campaign is divided into 3 phases: in the first one, an analysis of the CAN data is carried out to understand which of these features can be used in subsequent studies. The second part investigates the performance of the 3 strategies proposed in the section 6.3 to understand which one is better to exploit. Finally, different parameter configurations are explored to understand their impact on cluster results.

7.1 CAN-Bus Data Exploration

Given the problems reported in section 5.2.2 it is necessary to conduct a quantitative and qualitative analysis of the CAN data. During the analysis it must be understood how many vehicles have the *Connected* service, how many CAN headers are sent, how often and how much is the relationship with respect to the total messages. The analysis is conducted on the data relating to June 2020.

Of the 30,553 vehicles analyzed, only 4230 sent at least one CAN message, so only 13.84% of drivers use the Connected service. In total, the features that can be present within the various CAN messages, if we consider all the codings and protocols used, are 136, among these, 40 were selected as possibly useful. Within the June data, only 17 of the features considered interesting appear at least once, in general these features are very sparse, which increases the complexity of subsequent analyzes.

Usually a CAN message is sent together with a GPS message, so it is possible to approximate the average frequency to one minute. The messages containing at least one CAN header out of the total messages sent by the K-Master fleet are the 33.28%.

The table 7.1 shows the percentages of the frequencies of the CAN features that most often appear among the CAN messages.

Tag	Frequency in CAN msgs	Frequency in total msgs
FuelLevelPercentage	42.41%	14.11%
TimeEngineLife	40.19%	13.37%
TotalFuel	40.09%	13.34%
TotalVehicleDistance	40.06%	13.33%
EngineTemperature	25.67%	8.54%

Table 7.1: Percentage of CAN Data.

Among these five, the last one has values that are too low, it appears once in four in the CAN messages, to be used in the analysis. Only 4 features remain eligible, which in detail are:

- `FuelLevelPercentage`: is the ratio of the volume of fuel to the total volume of the fuel storage container. This feature, as interesting as it may seem, has not been used as it is common practice in trucks to add tanks that alter their value.

- `TimeEngineLife`: it is the accumulated time of operation of the engine. 2504 vehicles send this information, on average 34.00% of messages with CAN data of a file contain it.

- `TotalFuel`: it is the accumulated amount of fuel used during vehicle operation. 2899 vehicles send this information, on average 38.71% of messages with CAN data of a file contain it.

- `TotalVehicleDistance`: it is the accumulated distance traveled by the vehicle during its operation. 3574 vehicles send this information, on average 50.77% of messages with CAN data of a file contain it.

Through these analyzes, the minimum thresholds for the presence of CAN data within a trip have been defined so that the trip can be considered *Connected*.

7.2 Validation of the Strategies for Assigning the Type to Vehicles

In the registry the type of vehicle takes 297 different values but 60516 times its value is null, therefore the type of 92.89% of the vehicles in the fleet is not known. Figure 7.1 shows the most common types of vehicles and the number of times they appear in the registry.

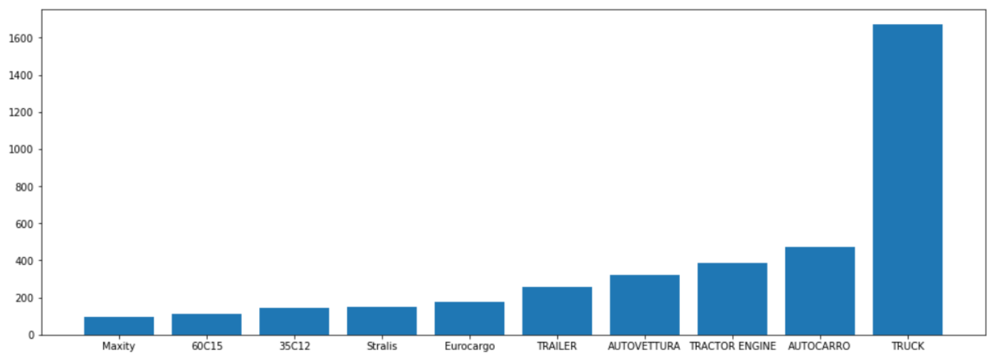


Figure 7.1: Histogram of the Most Common Vehicle Types.

Given the high presence of null values, in order to use the information regarding the type of vehicles, strategies for the automatic assignment of this information must be implemented. As reported in section 6.3, there are three strategies proposed: the first is based on a heuristic while the others are based on the application of classification algorithms. Considering the grouping defined in the section 6.3 we have identified 2333 HEAVY vehicles of which only 1638 have transmitted GPS data for the months of October 2019 and June 2020. As for the LIGHT vehicles, the numbers are slightly smaller, respectively 638 and 295 .

7.2.1 Heuristic Strategy

To test the validity of each threshold, the error produced was calculated by trying to categorize those vehicles whose type is known for sure thanks to the registry. So we used it as if it were a test set in a machine learning application. To avoid using wrong or uninteresting readings, all speeds above a certain threshold were limited to that value and all speeds below a second threshold were not considered.

In particular, the various heuristic thresholds selected and the related errors produced are:

- average speed = 50 km/h: with this threshold 85.54 % of HEAVY vehicles and 86.20 % of LIGHT vehicles are categorized incorrectly.

- average speed = 70 km/h: with this threshold, 33.14 % of HEAVY vehicles and 100 % of LIGHT vehicles are categorized incorrectly.
- maximum speed = 80 km/h: with this threshold, 95.43 % of HEAVY vehicles and 3.44 % of LIGHT vehicles are categorized incorrectly.
- maximum speed = 90 km/h: with this threshold 85.30 % of HEAVY vehicles and 17.24 % of LIGHT vehicles are categorized incorrectly.
- maximum speed = 100 km/h: with this threshold 28.10 % of HEAVY vehicles and 17.24 % of LIGHT vehicles are categorized incorrectly.

The results produced are not satisfactory, especially with regard to LIGHT vehicles. However, the best heuristic would seem to be based on a high maximum speed.

In figure 7.2 are shown the graphs of the average and maximum speeds of the vehicles. HEAVY ones are shown in red and LIGHT ones in blue. It is possible to notice how the values for the two categories are mixed, especially those concerning the average speed, which means that the application of a heuristic based purely on speed does not give sufficiently good results.

7.2.2 Machine Learning Strategy

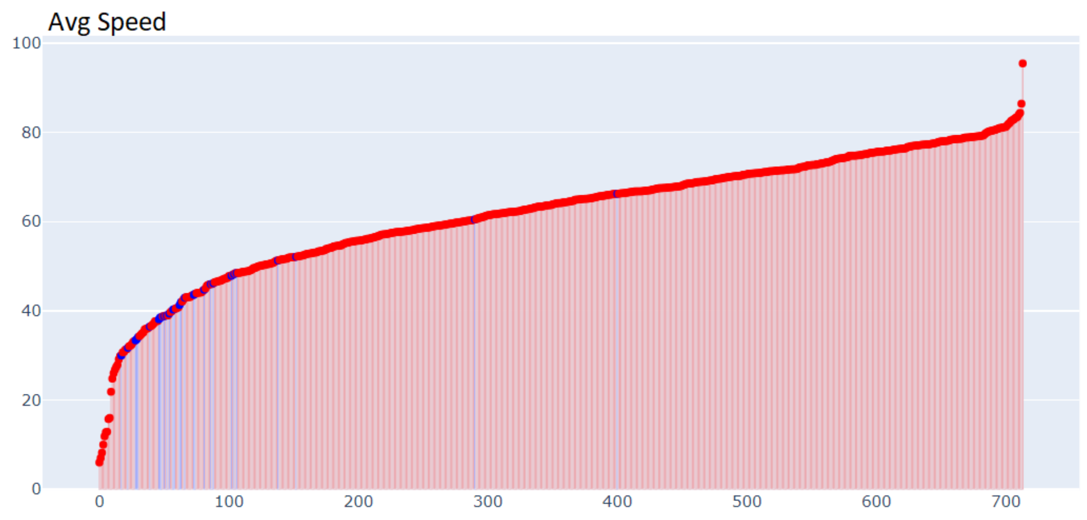
Two models are used to assign the type to the vehicles through machine learning algorithms: Random Forest (RF) and Gradient Boosting (GB) classifiers. Leave One Out CV was used to obtain more robust results. Accuracy, precision and recall for both classes were calculated to evaluate the two classifiers and understand which of the two performs better. The results obtained are in table 7.2.

	Accuracy	Precision H	Precision L	Recall H	Recall L
RF	89.30%	89.52%	80.48%	99.45%	16.09%
GB	89.30%	89.81%	73.58%	99.05%	19.02%

Table 7.2: Results obtained by the two algorithms.

Both classifiers achieve fairly high accuracy. As imaginable, precision and recall of the majority class (HEAVY) are also good for both. Unfortunately, it is more difficult to classify the minority class. In particular, it is possible to notice how both Random Forest and Gradient Boosting often predict this class even when it is not (Recall at 16.09% and 19.02%).

From the classifiers, it is possible to derive the feature importance given by the classification model. In figures 7.3 and 7.4 we can note that both algorithms do



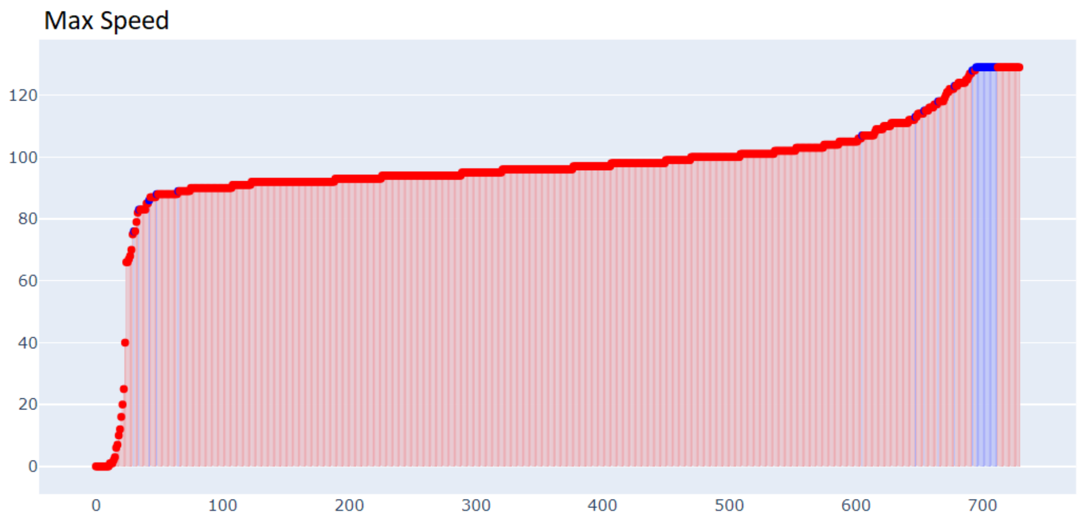


Figure 7.2: Distribution of Vehicle Speeds in the October 2019 Period.

not consider the features related to speed as the most important, confirming that the heuristics used cannot be applied.

Given the results obtained from the various proposed strategies, it was decided to use BG for the assignment of the vehicle type.

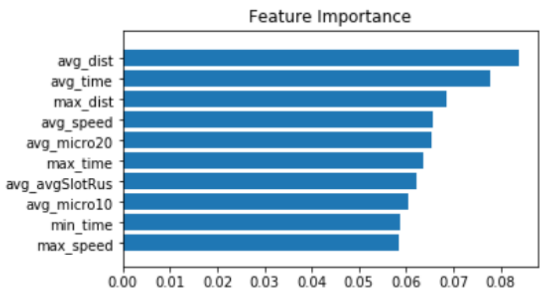


Figure 7.3: Features Importance for RF

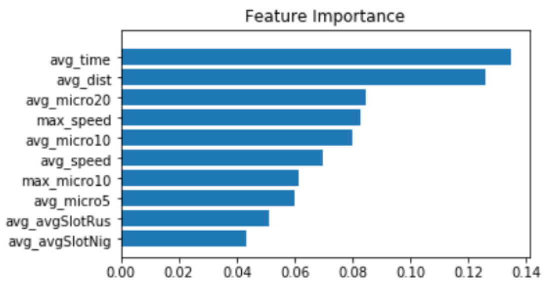


Figure 7.4: Features Importance for BG

7.3 Assigning Driver Behavior Experiments

In this phase we tested several configuration settings. Each of them corresponds to a combination of parameter values in the pipeline. For each configuration setting the clusters found are listed.

The experiments were run on the data collection relative to June 2020. To simulate real system usage, we split data into training and test sets. Specifically, trips performed between June 1, 2020 and June 22, 2020 were included in the training set and used to build the clusters. In the pipeline, the clustering results are used to calculate the KPI, which is based only on the *Connected* trips contained in the train set. For this reason, the *Business* trips contained in the train set and the trips contained in the test set are excluded from the analysis.

In the cluster experiments performed, we focused on the subset of routes associated with the highest number of *Connected* trips. To describe the output of the clustering algorithm we reported both the number of clusters and outliers found and the silhouette values. There is also a brief explanation of why outliers are defined as such.

7.3.1 Experiments setup

All data for the month of June were processed using the pipeline described in chapter 6 except for the step described in section 6.3. This is an optional step, so it can be omitted. The list of system parameters considered in our study is enumerated below.

- **Time delimiter (`td`).** The value of this parameter influences the trip definition. GPS messages are processed sequentially and their timestamp is used to group them on separate trips. If the elapsed time between two consecutive messages is greater than `td`, then the current trip is labeled as "concluded" and a new one is started.

- **Cell neighborhood size.** The size of the neighborhood affects the route identification phase. A large neighborhood entails including the adjacent cells to the starting and ending ones, resulting in larger source and destination areas.
- **Contextual features.** The availability of contextual features (e.g., altitude and meteorological conditions) influences the computation of the clusters.
- **`minPoints`.** It is the only parameter for the DBSCAN algorithm that must be set, ϵ is chosen accordingly with the technique illustrated in section 6.7.2. Its value is kept constant for all experiments at 10.

The metric used to calculate the distances between the points is the Minkowski distance.

In section 7.4 and 7.5 we report the results on several routes using as time delimiter values 20 and 40 minutes respectively.

7.4 Time delimiter: 20 minutes

This section shows the results achieved by setting the `td` to 20 minutes in the performed experiments. By processing all vehicle data with the aforesaid time delimiter we obtained 928,506 different trips belonging to 25,814 different vehicles. Routes identification on these trips is performed with several values of cell neighborhood size and for each identified route the DBSCAN is computed by both considering and not considering contextual features, listed in section 6.5.1.

7.4.1 Neighborhood size 0

Route: Marina Roseto Capo Spulico - Bari

The first route identified by setting the neighborhood size to zero has been traveled 65 times by 7 different vehicles. All the 65 trips belonging to the route are *Connected.* The path traveled by the trips of this route can be seen in figure 7.5.

The split between train and test set generated a train set composed of 46 trips and consequently a test set composed of 19 trips. Clustering produces one cluster if the contextual features are not considered and identifies a single outlier. If contextual features are introduced into the analysis, only one cluster is always identified, this time, however, two outliers are highlighted. In the first case, the behavior of the outlier is anomalous for the choice of the path and consequently of

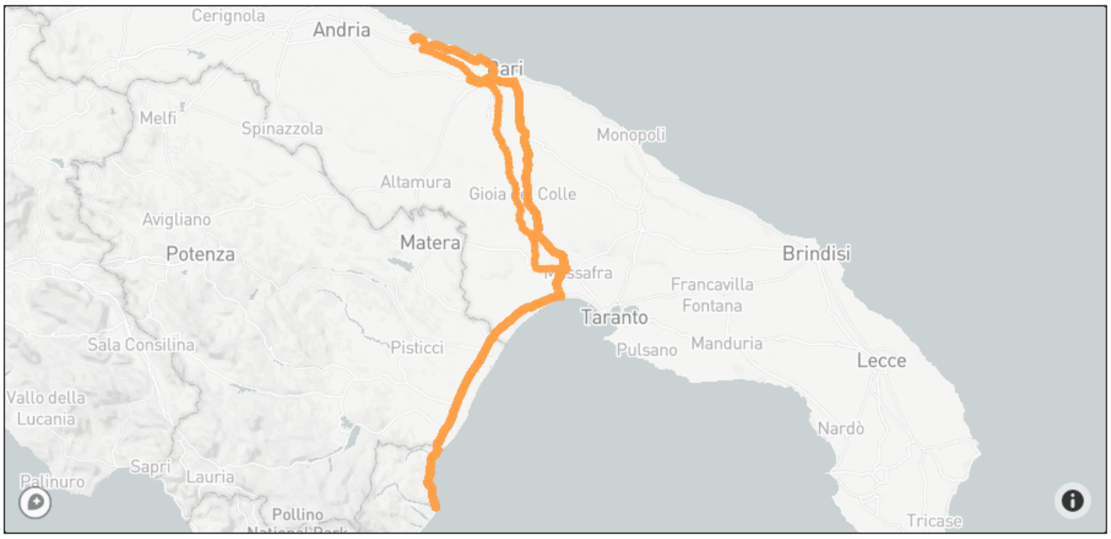


Figure 7.5: Map of the Marina Roseto Capo Spulico - Bari Route.

the features that concern it such as the time to complete it or the total distance. By introducing the contextual features the trip that was an outlier falls within the cluster. Two trips are excluded from this due to weather conditions, the two drivers maintained the same speeds despite the rain.

The silhouettes in the two cases are respectively 0.333 and 0.504, therefore the introduction of the contextual features allows to produce a more cohesive cluster with better separated outliers.

7.4.2 Neighborhood size 1

Route: Soccorso (PG) - Arezzo

The first route identified using a neighborhood size equals to one has been traveled 106 times by 28 different vehicles. Out of 106 trips, 57 of them are *Connected* and the remaining 49 trips are *Business*. The path traveled can be seen in figure 7.6.

The training set consists of 41 *Connected* trips and two test set, one made by 22 *Connected* trips and the other made by 8 *Business* trips. The remaining 22 *Business* trips traveled during the training phase were discarded since by construction the clustering phase exclusively relies on data acquired from *Connected* vehicles. Disregarding the contextual features, the clustering phase aggregated the trips into a cluster and found 8 outliers. Within them there are multiple behaviors, some of the most relevant features compared to the medoid are the travel times and the time slots in which the trip was conducted. Noteworthy is the behavior of an

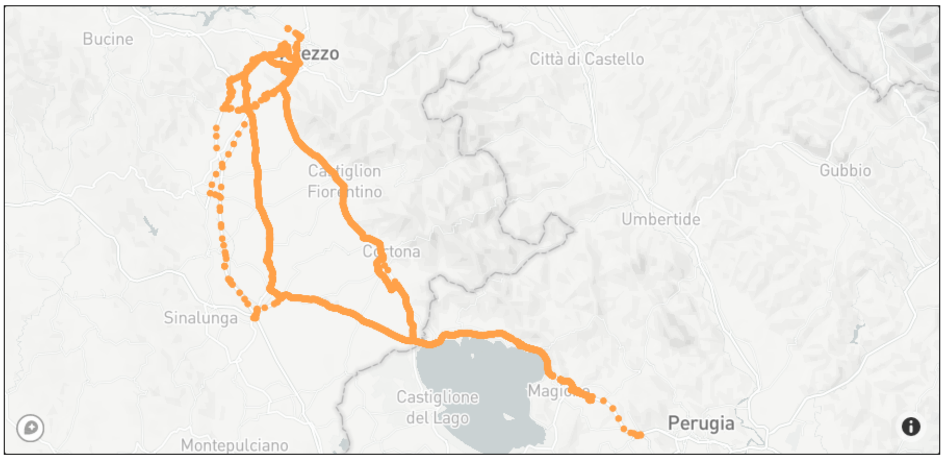


Figure 7.6: Map of the Soccorso (PG) - Arezzo Route.

outlier, whose average consumption is 60% compared to the majority cluster. The silhouette in this case is 0.343.

By introducing the weather features, a single cluster is formed, it includes all points except 11. The outliers of the previous case remain as such, to these are added those trips conducted in more adverse weather conditions. The silhouette is 0.199, this assumes a lower value than in the previous case. A decrease in this metric indicates that the points considered outliers only because of the contextual features have a fairly similar behavior with respect to the medoid and different with respect to the other outliers.

7.4.3 Neighborhood size 2

Route: Milano - Lodi

The first route identified using a neighborhood equals to two has been traveled 257 times by 121 different vehicles. Of the 257 trips, 113 are *Connected* whereas the remaining 144 are *Business*. The path traveled by the trips of this route can be seen in figure 7.7.

The split between train and test set generated a train set composed of 70 *Connected* trips and two test sets, one made by 43 *Connected* trips and the other made by 43 *Business* trips. 101 *Business* trips were excluded from the training set since they are not relevant to the clustering phase. The clustering phase aggregated the trips in a single cluster and detected 15 outliers, which were excluded from the train set. The group of outliers differs mainly for the average fuel consumption, for the time when the trips took place, in this case, the outliers made the trips at rush

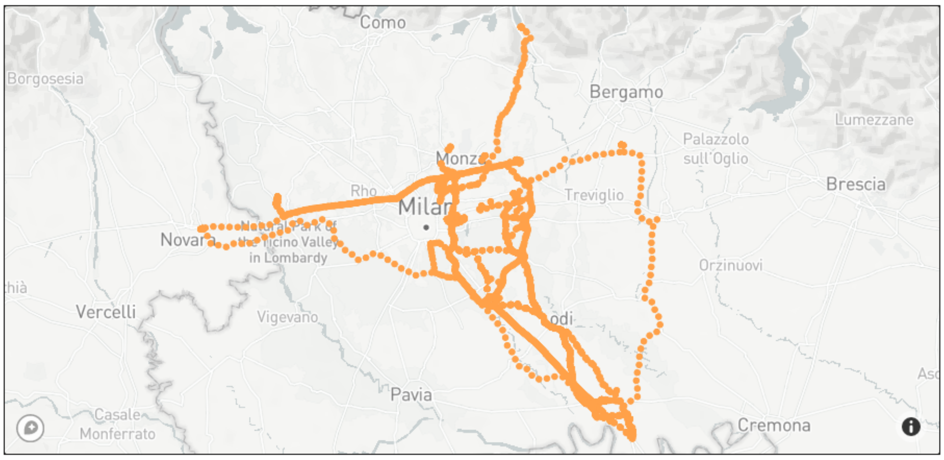


Figure 7.7: Map of the Milano - Lodi Route.

times, and for the chosen path. The addition of the contextual features creates always a cluster but only 4 outliers. Also in this case the outliers come back to the cluster because the trips were conducted under similar weather conditions. The remaining outliers differ from the cluster due to the features concerning both the possibility of precipitation and the time taken to make the trip. Silhouettes take 0.315 and 0.395 as values.

7.5 Time delimiter: 40 minutes

This section summarizes the results achieved by setting the time delimiter to 40 minutes. Processing all vehicle data with the aforesaid value of time delimiter we obtained 652957 different trips belonging to 26749 different vehicles.

As in the experiments of the previous chapter, the following sections show the most traveled routes calculated with different neighborhood values. For each of them, the clusters are obtained both with contextual features and without them.

7.5.1 Neighborhood size 0

Route: Marina Roseto Capo Spulico - Bari

The first route identified has been traveled 62 times by 7 different vehicles. A subset of these trips is trips analyzed in section 7.4.1, as in that case all of them are *Connected.*

The division between train and test set generates two groups of 43 and 19 trips.

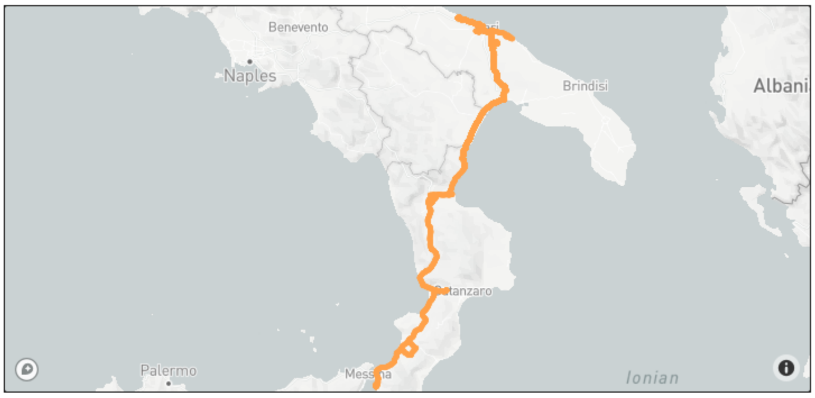


Figure 7.8: Map of the Marina Roseto Capo Spulico - Bari Route.

The clustering application generates, both in the case in which the contextual functionalities are used and in the unused case, a single cluster. The anomalous values detected are always respectively 2. If the contextual features are disabled, the outliers have an average difference in fuel consumption greater than 43% compared to the medoid, moreover one of the two outliers also has a higher average speed of 40% to the medoid. If the contextual features are enabled, the latter outlier comes back to the cluster and one trip that was previously there leaves it. This last trip also differs in higher speeds and shorter travel times than the new medoid, plus the weather conditions with which the trips were done are decidedly more adverse than the medoid ones.

Once again, introducing the contextual features, the silhouette increases, passing from 0.496 to 0.528. These values are higher than those reported in section 7.4.1, which could lead to believe that the new trips are more homogeneous than those identified with `td` = 20. If contextual features are excluded, from a more accurate analysis, we see that one of the two outliers is a trip that makes a large detour in southern Italy and that therefore, in the space in which the trips are represented, is very distant from the medoid. The mean of the distance between the outliers and the cluster points is much greater than the mean distance between the cluster points. This causes the silhouette to grow despite the fact that most of the trips have remained the same and that the trip has been introduced which should not be considered in this route because it has a completely different path. For this section, therefore, it is advisable to use `td`= 20.

7.5.2 Neighborhood size 1

Route: Soccorso (PG) - Arezzo

As we can see from the name of the route, the first route identified using a neighborhood equals one has the start and end points equals to the example 7.4.2. However in this case the route is traveled 98 times by 25 vehicles, the decrease in the number of trips is due to the change of the `td` parameter. In fact, some trips depart from Soccorso, go to Arezzo and make a break there longer than 20 minutes but less than 40, to then reach other destinations. Such trips are included in the analysis reported in section 7.4.2 but not in this one. The set of trips is made up of 50 *Connected* and 48 *Business*.

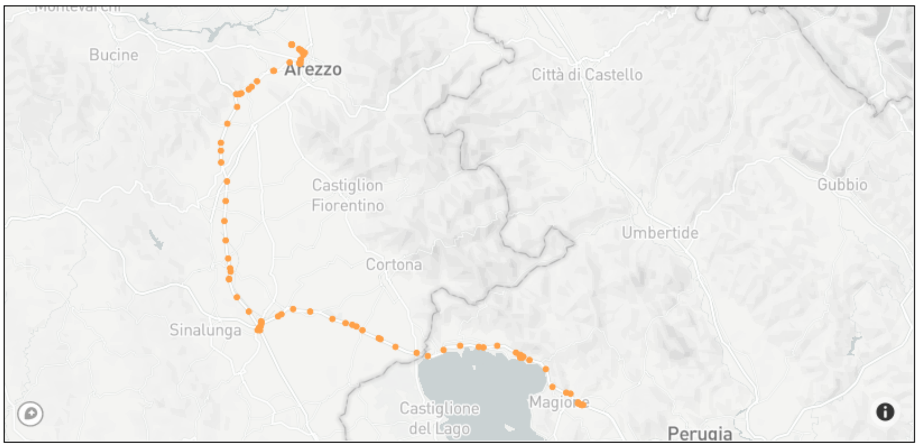


Figure 7.9: Map of the Soccorso (PG) - Arezzo Route.

The division between train and test set generated a train set consisting of 37 *Connected* trips and two test sets, one consisting of 19 *Connected* and the other of 9 *Business*. The 33 *Business* trips made before the split date were discarded and not taken into account during the cluster phase. Only one cluster is formed excluding the contextual characteristics while the outliers are 8. Also in this case some trips are outliers because they have made different paths to respect the medoid. The introduction of the contextual features causes the algorithm to generate only one cluster and highlight only 10 outliers. In general, the outliers make fewer pauses within the trip. In particular, one outlier has the average speed higher than the medoid of 30% while two others conducted the trip with the same speed but with a visibility of the 50% lower than the medoid.

The silhouettes obtained are 0.324 and 0.199, these are very similar to those calculated with `td` = 20. This indicates that the excluded trips, using `td` = 40, were homogeneous with the elements of the groups, clusters and outliers, to which they belonged.

7.5.3 Neighborhood size 2

Route Soccorso (PG) - Arezzo

Once again the route with more *Connected* trip starts from Soccorso and arrives in Arezzo, also in this case by increasing the value of the neighborhood more trips are incorporated into the analysis. Indeed this time the route has been traveled 106 times, 51 times by *Connected* trips and the remain by *Business* trips, by 27 different vehicles.

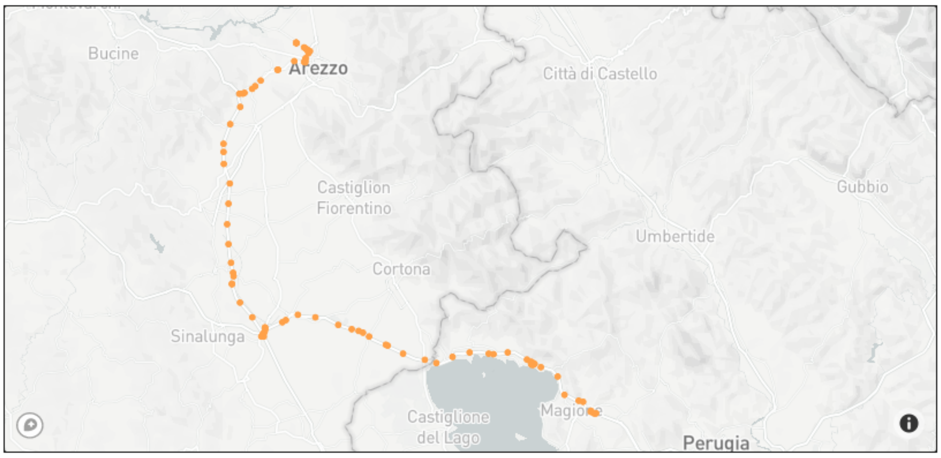


Figure 7.10: Map of the Route: Soccorso (PG) - Arezzo Route.

The split between train and test set generated a train set composed of 57 *Connected* trips and two test sets, one made by 13 *Connected* and the other made by 9 *Business*. The 27 *Business* trips traveled before the split date is discarded and not considered during the cluster calculation. So the ones that were used to perform the analysis are all those of the previous route except for one that is added to the train set. Excluding the contextual features, one cluster and 8 outliers emerge, these are the same trips obtained with neighborhood = 1, therefore the increase of the neighborhood introduces in the section a homogeneous trip with respect to the majority of the elements contained in it. Even considering the contextual features, the new trip remains within the cluster, its introduction does not change the number of outliers, which therefore remain 10. The reasons why the outliers are considered anomalous are the same as in the previous case. The calculated silhouettes are 0.343 and 0.197. If the contextual features are not considered, the insertion of the new point allows the creation of slightly more cohesive clusters. The difference between the values of the silhouette reported in the section 7.5.2 is just 0.019. The introduction of contextual features, on the other hand, does not change the cohesion of the clusters compared to the previous experiment

Chapter 8

Conclusions and Future Developments

This thesis aims to create a highly customizable framework capable of identifying, characterizing and separating different driving styles by merging multiple raw data sources. Position, state and type of the vehicle and weather conditions in which the trip was conducted are the information gathered.

The methodology used to determine driving behaviors takes inspiration from several of the works found in the literature by combining the various approaches seen in the section 4 in a single pipeline. Contrary to most of these works, in which the aim is generally to determine whether a driving style is correct or not, in the developed work the number of clusters contained in a section is not known a priori and therefore we do not know if they will be present only two types of behavior.
Another distinction with the standard approach for driving behavior investigation was the usage of different kinds of data, listed in section 5. It allowed us to examine those cases in which the CAN features are not present in a sufficient number. The strong lack of this data means that the analyzes were carried out in an adverse context, information regarding the state of the vehicle did not guide the exploration of the data as is the case for most of the related works.

Various considerations are evident from the experimental campaign, which can divide according to the phase concerning:

- first phase: given the data received from K-Master, it is evident that most of the information collected does not have or only has a part of the CAN features. In subsequent analyzes, it has been impossible to use a lot of types of information regarding the state of the vehicle.
- second phase: results obtained show that it is not possible to assign the type,

which is a relatively easy task compared to determining the driving behavior, of a vehicle based only on speed. The vehicle speed is not only linked to the type of the vehicle but also the road it travels on. It reinforces our idea that, to determine driving behavior, it is not enough to observe the data relating to the driver/vehicle but also data describing the environment should be used.

- third phase: experiments related to the clustering phase show the performance of the framework with various parameter configurations. Trips were defined using a different `td`, routes were identified with different neighbors values and clustering was calculated with both the contextual features than without. So we can draw conclusions for each of these parameters.
  - *Time delimiter*: increasing it can lead to longer trips within the routes. On one hand, it means that the breaks, perhaps due to the loading/unloading of goods, do not divide the sequence of GPS points into more trips when in reality the trip is unique. On the other hand, can bring to the creation of routes containing heterogeneous trips, distorting the analyzes. In the examples in section 7.4.1 and section 7.5.1 this happens. In the case of `td` = 40, a vehicle is reported as an outlier while it was contained in the cluster in the case of `td` = 20. It happens because this vehicle did not initially go towards Bari but made a detour to southern Italy. This deviation, which contains a break greater than 20 minutes but less than 40, modifies all the features inherent to the route taken, such as the time taken to make the trip or the height difference, thus creating a trip with anomalous values compared to the other trips of the route.
  - *Neighborhood*: changing it increases the number of trips contained within a route, thus changing its importance in the ranking. In the examples given in section 7.5.2 and section 7.5.3 the most important route remains the same but increases the number of its trips. In this case, the trip that remains excluded with `neighborhood` = 1 travels the same path as the other trips, so it is correct to include it in the analysis.
  - *Contextual features*: the use of them strongly influences the outcome of clustering. The experiments show that if a trip is made in the rain or on a cloudy day, the driver's behavior is abnormal. It is because all the trips were made in June in Italy and therefore the driver behavior is not necessarily anomalous but rather the weather conditions in which the trip was made are anomalous. Then if on one hand, the contextual features allow us to better understand the driver behavior by describing, although partially, the environment in which the data is collected, on the other hand, if the distribution of these features is strongly unbalanced, their introduction risks compromising the analysis. This risk can be eliminated by

analyzing data over a larger period in such a way that the weather conditions in which the trips are conducted do not take on practically a single value. Another applicable strategy, to remove this problem, would be to group data not only by route but also according to weather conditions. However, to do this would have needed much more data.

Future work can improve the accuracy of the analysis. A possible strategy could be to change the concept of route, defining it no longer based on a pair of cells but based on the path followed. In this way, we avoid reporting as anomalous a trip that followed a different road than the centroid of that route. It was impossible to implement in the thesis as the scarcity of data would have led to sections containing too few trips for analysis. With the increase in the amount of data available, it would also be possible to apply Deep Learning strategies. In the latter case, techniques that allow incremental training of the model are certainly useful. It would also be interesting to study the application of other density-based clustering techniques capable of determining clusters with different densities as OPTICS and HDBSCAN.

Bibliography

[1] Kawtar ZINEBI, Nissrine SOUISSI, and Kawtar TIKITO. «Driver Behavior Analysis Methods: Applications oriented study». In: () (cit. on p. 5).

[2] Michał Czubenko, Zdzisław Kowalczuk, and Andrew Ordys. «Autonomous driver based on an intelligent system of decision-making». In: *Cognitive computation* 7.5 (2015), pp. 569–581 (cit. on p. 5).

[3] Mario Gerla, Eun-Kyu Lee, Giovanni Pau, and Uichin Lee. «Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds». In: *2014 IEEE world forum on internet of things (WF-IoT)*. IEEE. 2014, pp. 241–246 (cit. on p. 5).

[4] C Prabha, R Sunitha, and R Anitha. «Automatic vehicle accident detection and messaging system using GSM and GPS modem». In: *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering* 3.7 (2014), pp. 10723–10727 (cit. on p. 5).

[5] Syedul Amin, Mamun Bin Ibne Reaz, and Salwa Sheikh Nasir. «Integrated vehicle accident detection and location system». In: *Telkomnika* 12.1 (2014), p. 73 (cit. on p. 5).

[6] Mikko Perttunen et al. «Distributed road surface condition monitoring using mobile phones». In: *International conference on ubiquitous intelligence and computing*. Springer. 2011, pp. 64–78 (cit. on p. 5).

[7] Ravi Bhoraskar, Nagamanoj Vankadhara, Bhaskaran Raman, and Purushottam Kulkarni. «Wolverine: Traffic and road condition estimation using smartphone sensors». In: *2012 fourth international conference on communication systems and networks (COMSNETS 2012)*. IEEE. 2012, pp. 1–6 (cit. on p. 5).

[8] Derick A Johnson and Mohan M Trivedi. «Driving style recognition using a smartphone as a sensor platform». In: *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2011, pp. 1609–1615 (cit. on p. 6).

[9] Geqi Qi, Yiman Du, Jianping Wu, and Ming Xu. «Leveraging longitudinal driving behaviour data with data mining techniques for driving style analysis». In: *IET intelligent transport systems* 9.8 (2015), pp. 792–801 (cit. on p. 6).

[10] Jaeik Jo, Sung Joo Lee, Kang Ryoung Park, Ig-Jae Kim, and Jaihie Kim. «Detecting driver drowsiness using feature-level fusion and user-specific classification». In: *Expert Systems with Applications* 41.4 (2014), pp. 1139–1152 (cit. on p. 6).

[11] Ines Teyeb, Olfa Jemai, Mourad Zaied, and Chokri Ben Amar. «A novel approach for drowsy driver detection using head posture estimation and eyes recognition system based on wavelet network». In: *IISA 2014, The 5th International Conference on Information, Intelligence, Systems and Applications*. IEEE. 2014, pp. 379–384 (cit. on p. 6).

[12] P. Ping, W. Qin, Y. Xu, C. Miyajima, and K. Takeda. «Impact of Driver Behavior on Fuel Consumption: Classification, Evaluation and Prediction Using Machine Learning». In: *IEEE Access* 7 (2019), pp. 78515–78532. DOI: 10.1109/ACCESS.2019.2920489 (cit. on p. 6).

[13] Javier E Meseguer, Carlos T Calafate, Juan Carlos Cano, and Pietro Manzoni. «Assessing the impact of driving behavior on instantaneous fuel consumption». In: *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*. IEEE. 2015, pp. 443–448 (cit. on p. 6).

[14] C. D'Agostino, A. Saidi, G. Scouarnec, and L. Chen. «Rational truck driving and its correlated driving features in extra-urban areas». In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. 2014, pp. 1199–1204. DOI: 10.1109/IVS.2014.6856440 (cit. on p. 6).

[15] David L Armitage, Gregory Froim Kushnir, and Mark Alvin Mason. *Driver performance analysis and consequence*. US Patent 9,082,308. July 2015 (cit. on p. 6).

[16] Leonard Evans and Paul Wasielewski. «Risky driving related to driver and vehicle characteristics». In: *Accident Analysis & Prevention* 15.2 (1983), pp. 121–136 (cit. on pp. 6, 7).

[17] S. Kaplan, M. A. Guvensan, A. G. Yavuz, and Y. Karalurt. «Driver Behavior Analysis for Safe Driving: A Survey». In: *IEEE Transactions on Intelligent Transportation Systems* 16.6 (2015), pp. 3017–3032. DOI: 10.1109/TITS.2015.2462084 (cit. on p. 7).

[18] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. «Supervised machine learning: A review of classification techniques». In: *Emerging artificial intelligence applications in computer engineering* 160.1 (2007), pp. 3–24 (cit. on p. 12).

[19] Donald Michie, David J Spiegelhalter, and Charles C Taylor. «Machine learning, neural and statistical classification». In: (1994) (cit. on p. 12).

[20] A. Veloso, W. Meira, and M. J. Zaki. «Lazy Associative Classification». In: *Sixth International Conference on Data Mining (ICDM'06)*. 2006, pp. 645–654. DOI: `10.1109/ICDM.2006.96` (cit. on p. 12).

[21] Prafulla B Bafna and Jatinderkumar R Saini. «On Exhaustive Evaluation of Eager Machine Learning Algorithms for Classification of Hindi Verses». In: *International Journal of Advanced Computer Science and Applications* (2020) (cit. on p. 12).

[22] S. R. Safavian and D. Landgrebe. «A survey of decision tree classifier methodology». In: *IEEE Transactions on Systems, Man, and Cybernetics* 21.3 (1991), pp. 660–674. DOI: `10.1109/21.97458` (cit. on p. 13).

[23] Mr. Brijain, R Patel, Mr. Kushik, and K Rana. *A Survey on Decision Tree Algorithm For Classification* (cit. on p. 13).

[24] F. Esposito, D. Malerba, G. Semeraro, and J. Kay. «A comparative analysis of methods for pruning decision trees». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19.5 (1997), pp. 476–491. DOI: `10.1109/34.589207` (cit. on p. 14).

[25] W. N. H. W. Mohamed, M. N. M. Salleh, and A. H. Omar. «A comparative study of Reduced Error Pruning method in decision tree algorithms». In: *2012 IEEE International Conference on Control System, Computing and Engineering*. 2012, pp. 392–397. DOI: `10.1109/ICCSCE.2012.6487177` (cit. on p. 14).

[26] Thomas G Dietterich et al. «Ensemble learning». In: *The handbook of brain theory and neural networks* 2 (2002), pp. 110–125 (cit. on p. 14).

[27] Omer Sagi and Lior Rokach. «Ensemble learning: A survey». In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8.4 (2018), e1249 (cit. on p. 14).

[28] Leo Breiman. *Bias, variance, and arcing classifiers.* Tech. rep. Tech. Rep. 460, Statistics Department, University of California, Berkeley ..., 1996 (cit. on p. 15).

[29] Harris Drucker and Corinna Cortes. «Boosting Decision Trees.» In: vol. 8. Jan. 1995, pp. 479–485 (cit. on p. 15).

[30] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. «LightGBM: A Highly Efficient Gradient Boosting Decision Tree». In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017, pp. 3146–3154. URL: `https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf` (cit. on p. 15).

[31] K Kameshwaran and K Malarvizhi. «Survey on clustering techniques in data mining». In: *International Journal of Computer Science and Information Technologies* 5.2 (2014), pp. 2272–2276 (cit. on p. 15).

[32] R. Indhu and R. Porkodi. «Comparison of Clustering Algorithm». In: *International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT)* 3.1 (2018), pp. 218–223 (cit. on pp. 17, 20).

[33] Adam Chehouri, Rafic Younes, Jihan Khoder, Jean Perron, and Adrian Ilinca. «A Selection Process for Genetic Algorithm Using Clustering Analysis». In: *Algorithms* 10.4 (). DOI: `10.3390/a10040123(2017)` (cit. on p. 17).

[34] X. Wang, T. Zhang, and X. Gao. «Multiview Clustering Based on Non-Negative Matrix Factorization and Pairwise Measurements». In: *IEEE Transactions on Cybernetics* 49.9 (2019), pp. 3333–3346. DOI: `10.1109/TCYB.2018.2842052` (cit. on p. 17).

[35] A. O. Aseeri, Y. Zhuang, and M. S. Alkatheiri. «A Memory Capacity-Aware Algorithm for Fast Clustering of Disk-Resident Big Datasets». In: *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*. 2017, pp. 194–201. DOI: `10.1109/DASC-PICom-DataCom-CyberSciTec.2017.44` (cit. on p. 18).

[36] Pranav Nerurkar, Archana Shirke, Madhav Chandane, and Sunil Bhirud. «Empirical Analysis of Data Clustering Algorithms». In: *Procedia Computer Science* 125 (Jan. 2018), pp. 770–779. DOI: `10.1016/j.procs.2017.12.099` (cit. on p. 18).

[37] Joerg Sander. «Density-Based Clustering». In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 270–273. ISBN: 978-0-387-30164-8. DOI: `10.1007/978-0-387-30164-8_211`. URL: `https://doi.org/10.1007/978-0-387-30164-8_211` (cit. on p. 19).

[38] Pradeep Rai and Shubha Singh. «A Survey of Clustering Techniques». In: *nternational Journal of Computer Applications (* 7.12 (2010) (cit. on p. 19).

[39] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. «OPTICS: Ordering points to identify the clustering structure». In: *ACM Sigmod record* 28.2 (1999), pp. 49–60 (cit. on pp. 19, 26).

[40] Ricardo JGB Campello, Davoud Moulavi, and Jörg Sander. «Density-based clustering based on hierarchical density estimates». In: *Pacific-Asia conference on knowledge discovery and data mining*. Springer. 2013, pp. 160–172 (cit. on pp. 19, 27).

[41] Nadav Bar, Hadar Averbuch-Elor, and Daniel Cohen-Or. «Border-peeling clustering». In: *arXiv preprint arXiv:1612.04869* (2016) (cit. on pp. 19, 27).

[42] Xiang Li, Ben Kao, Caihua Shan, Dawei Yin, and Martin Ester. *CAST: A Correlation-based Adaptive Spectral Clustering Algorithm on Multi-scale Data*. 2020. arXiv: `2006.04435 [cs.LG]` (cit. on p. 20).

[43] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. «A density-based algorithm for discovering clusters in large spatial databases with noise». In: AAAI Press, 1996, pp. 226–231 (cit. on pp. 20, 21, 24, 50).

[44] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. «DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN». In: 42.3 (July 2017). ISSN: 0362-5915. DOI: `10.1145/3068335`. URL: `https://doi.org/10.1145/3068335` (cit. on pp. 21, 24).

[45] T. Ali, S. Asghar, and Naseer Ahmed Sajid. «Critical analysis of DBSCAN variations». In: *2010 International Conference on Information and Emerging Technologies*. 2010, pp. 1–6. DOI: `10.1109/ICIET.2010.5625720` (cit. on p. 24).

[46] Y. Wu, J. Guo, and X. Zhang. «A Linear DBSCAN Algorithm Based on LSH». In: *2007 International Conference on Machine Learning and Cybernetics*. Vol. 5. 2007, pp. 2608–2614. DOI: `10.1109/ICMLC.2007.4370588` (cit. on p. 24).

[47] B. Liu. «A Fast Density-Based Clustering Algorithm for Large Databases». In: *2006 International Conference on Machine Learning and Cybernetics*. 2006, pp. 996–1000. DOI: `10.1109/ICMLC.2006.258531` (cit. on p. 24).

[48] Hongquan Jiang, Wang Rongxi, Jianmin Gao, Zhiyong Gao, and Xu Gao. «Evidence Fusion-Based Framework for Condition Evaluation of Complex Electromechanical System in Process Industry». In: *Knowledge-Based Systems* 124 (May 2017), pp. 176–187. DOI: `10.1016/j.knosys.2017.03.011` (cit. on p. 25).

[49] Amin Karami and Ronnie Johansson. «Choosing DBSCAN Parameters Automatically using Differential Evolution». In: *Int. J. Comput. Appl.* 91 (Mar. 2014). DOI: `10.5120/15890-5059` (cit. on p. 25).

[50] W. Lai, M. Zhou, F. Hu, K. Bian, and Q. Song. «A New DBSCAN Parameters Determination Method Based on Improved MVO». In: *IEEE Access* 7 (2019), pp. 104085–104095. DOI: `10.1109/ACCESS.2019.2931334` (cit. on p. 25).

[51] A. Smiti and Z. Elouedi. «DBSCAN-GM: An improved clustering method based on Gaussian Means and DBSCAN techniques». In: *2012 IEEE 16th International Conference on Intelligent Engineering Systems (INES)*. 2012, pp. 573–578. DOI: `10.1109/INES.2012.6249802` (cit. on p. 26).

[52] Zohreh Akbari and Rainer Unland. «Automated determination of the input parameter of dbscan based on outlier detection». In: *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer. 2016, pp. 280–291 (cit. on p. 26).

[53] Shi-Huang Chen, Jeng-Shyang Pan, and Kaixuan Lu. «Driving Behavior Analysis Based on Vehicle OBD Information and AdaBoost Algorithms». In: *IMECS* (2015) (cit. on p. 28).

[54] H. Eren, S. Makinist, E. Akin, and A. Yilmaz. «Estimating driving behavior by a smartphone». In: *2012 IEEE Intelligent Vehicles Symposium*. 2012, pp. 234–239. DOI: `10.1109/IVS.2012.6232298` (cit. on p. 28).

[55] G. Li, F. Zhu, X. Qu, B. Cheng, S. Li, and P. Green. «Driving Style Classification Based on Driving Operational Pictures». In: *IEEE Access* 7 (2019), pp. 90180–90189. DOI: `10.1109/ACCESS.2019.2926494` (cit. on pp. 28, 29).

[56] P. Brombacher, J. Masino, M. Frey, and F. Gauterin. «Driving event detection and driving style classification using artificial neural networks». In: *2017 IEEE International Conference on Industrial Technology (ICIT)*. 2017, pp. 997–1002. DOI: `10.1109/ICIT.2017.7915497` (cit. on p. 29).

[57] Jae-Gil Lee, Jiawei Hans, and Kyu-Young Whang. «Trajectory Clustering: A Partition-and-Group Framework». In: () (cit. on p. 29).

[58] Andrey Tietbohl, Vania Bogorny, Bart Kuijpers, and Luis Otavio Alvares. «A Clustering-based Approach for Discovering Interesting Places in Trajectories». In: () (cit. on p. 29).

[59] C. Marks, A. Jahangiri, and S. G. Machiani. «Iterative DBSCAN (I-DBSCAN) to Identify Aggressive Driving Behaviors within Unlabeled Real-World Driving Data». In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. 2019, pp. 2324–2329. DOI: `10.1109/ITSC.2019.8917162` (cit. on p. 30).

[60] Kuan-Ting Chen and Winnie Chen. «Driving Style Clustering using Naturalistic Driving Data». In: *Transportation Research Record: Journal of the Transportation Research Board* 2673 (May 2019), p. 036119811984536. DOI: `10.1177/0361198119845360` (cit. on p. 30).

[61] B. Higgs and M. Abbas. «A two-step segmentation algorithm for behavioral clustering of naturalistic driving styles». In: *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. 2013, pp. 857–862. DOI: `10.1109/ITSC.2013.6728339` (cit. on p. 30).

[62] Josh Lipkowitz and Vadim Sokolov. «Clusters of Driving Behavior From Observational Smartphone Data». In: *IEEE Intelligent Transportation Systems Magazine* PP (Oct. 2017). DOI: `10.1109/MITS.2019.2919516` (cit. on p. 30).

[63] *Dark Sky API website*. URL: `https://darksky.net/dev/` (cit. on p. 34).

[64] *Jawg API website*. URL: `https://www.jawg.io/docs/` (cit. on p. 35).

[65] *OnWater website*. URL: `https://onwater.io/` (cit. on p. 41).

[66] *Locationiq website*. URL: `https://locationiq.com/` (cit. on p. 48).