POLITECNICO DI TORINO

Master Degree in Mathematical Engineering

Master Thesis

Machine Learning for crash event detection from black-box data

Relatore:
Prof. Giacomo Como

Candidato:
Domenica Chiara Verbaro

Accademic year 2017-2018

Contents

Abstract

This thesis focuses on the statistical analysis of data collected by black boxes in order to detect car crash events. These boxes are installed on private vehicles and are equipped with GPS, accelerometer, and gyroscope. First, unsupervised machine learning and clustering algorithms will be applied to analyse the data and explore possible underlying structure. Then supervised statistical learning techniques will be applied to develop a classifier that is able to predict whether a crash has occurred or not. Finally we will try to train and test the best models found in the second phase to data divided into cluster, in order to see if we can improve our prevision.

Chapter 1

Introduction

Nowadays, huge amounts of data are collected and analyzed by industrial, commercial, and services companies. These data serve for different purposes such as organizing marketing strategies, minimizing costs, predicting high demand of goods, financial fluctuations , etc. For these reasons, large resources are invested in the gathering, storage, and analysis of big data, using machine learning techniques.

Machine learning algorithms use mathematical methods to perform data analysis. Some of these machine learning algorithms, called supervised methods, can be used in order to predict a quantity, while there are some other unsupervised techniques that can split our data into clusters. In the former case, we consider both the attributes that identify the characteristics of samples, both the information relative to the quantity that we want to predict. These algorithms find a decision rule or a decision function that links each observation to a value for target variable. In the latter situation, we do not take into account any label, but we just want to find out some hidden pattern. Moreover we could also combine different types of algorithms, improving the statistical learning models obtained.

This work deals with the application of machine learning techniques to a data-set generated by measurements recorded by the black-boxes installed on cars and motorbikes by the insurance company *UnipolSai*. In this situation, data analysis is performed in order to classify the signals into two categories: false alarms and real crashes.

UnipolSai is an Italian insurance company, world leader in car insurance telematics. Indeed it has over 2.5 billions of customers who decided to install black boxes. UnipolSai invested in *Alfaevolution technology*, a company whose goal is to provide services for car, house, health and well-being. As stated in its website, the purpose of *Alfaevolution technology* is to guarantee to the customers of UnipolSai, services for security and the simplification of the procedure after a crash. In the next paragraphs, we will describe how the black boxes installed by *Alfaevolution technology* work.

A Black box is an electronic device that collects information about the use and the position of the vehicle on which it is installed. In case of accident, the elaboration and the analysis of the black box data allows one to rebuild objectively the dynamic of an event. The main advantages of the black boxes, from the customers' perspective, are that the GPS allows a continuous localization of the vehicle and guarantees 24hours assistance in case of accident. Moreover people who decide to install these devices can obtain some sales on the cost of the insurance.

On the other hand, the main problem of the black boxes offers is how to identify if the signal provided refers to a real crash or a false alarm. The operation center of the company collects the data during different time intervals for each component: GNSS module records data from about 30 seconds/1 minute before and after the crash , while the acceleromter sends data 6/8 seconds before and after the events. All these signals are preprocessed and merged together, generating one single sample for each accident. After that, Alfaevoultion technology operators derive some key performance indicators(KPIs), using the data provided by accelerometer and GNSS module.

At the moment, the algorithm used by the company operators to predict whether an alarm identifies a crash is based on the comparison and the measurement of some physical quantities, while in this work we will focus on the derivation of an new algorithm using machine learning techniques. For instance a crash is defined as an event during which the value of acceleration/deceleration remains higher than twice the gravity acceleration(g) for a certain interval of time. On the other hand a mini-crash occurs when acceleration values are between once or twice the gravity acceleration for a certain time lag, during this period is possible to register some acceleration

greater than 2g, but the interval during which high acceleration is detected is too small to classify the event as a crash.

1.1 Purpose of this work

In this work we will implement a model able to classify alarms into real crashes or false accidents, using data provided by *Alfaevolution technology*. We will deal not with the original data generated by the devices installed on the vehicles, but only with some of the KPIs calculating by Alfaevoultion operators. The main goals of this work are:

1. Analyze attributes distributions;
2. Identify relationships between features;
3. Try to reduce the dimensionality of the data set;
4. Look for noticeable patterns that characterize false and real crashes;
5. Fit a supervised classifier that predict the highest number of correct labels for each sample;
6. Try to reduce the number of misclassified false alarms, in order to limit costs.

After a brief description of the data set(see Chapter 2), we will try to understand more in depth the distribution of the KPIs, calculating the correlation between them and trying to detect the outliers present among the data, using isolation forests. Then we will do an unsupervised study of the samples, applying two different strategies: cluster analysis and manifold learning. The former consists in dividing observations into groups, in order to identify some similarities among the data and it will be used to check if the samples referring to false crashes and real accidents can be easily divided. On the other hand, the latter strategy will be applied in order to reduce the number of features composing our data set, projecting the samples in a lower-dimensional space. Finally we will use different supervised learning algorithms to fit a model that classifies alarms with a higher precision than the one of Alfaevolution classifier.

Chapter 2

Data set description

In this chapter, we will briefly describe the black boxes mechanism, and then we will start to explore the data set provided in order to understand distributions and characteristics of the samples.

2.1 Black boxes

As explained in the Introduction, black boxes are devices used in order to record data about the state of the vehicle on which they are installed. The space occupied by a black box is very small, in fact it is more or less as big as a smartphone. Modern models can be used as anti-theft alarms, but they have to be installed by a professional in a non visible part of the car, such as the engine compartment, while other basic models can be auto-installed directly by the customers on the battery of the cars. Some of these recent devices offer more services to the customers, that can be used everyday, even while using the car. In Figure 2.1 it is reported one of the black boxes installed. The main components of the black boxes are:

- a GSM module, that sends information to the operation center;
- a GPS component, that ensures the geolocalization of the vehicle;
- one or more accelerometers, that measure the forces acting on the vehicle;
- a microporcessor with physical memory, that elaborates the data.

Figure 2.1: An example of black box

Looking at the data emitted by a black box, it is possible to understand the actual accident dynamics, from seconds before the impact, to some time after the crash, even if there are no witnesses. The black boxes are always active and it can be questioned by the operation center when the client requires it(as in case of a theft), both directly by the owner of the car using a smartphone application. Moreover, as soon as these devices detect a crash or a mini-crash(such as impact with a stationary object) it sends a signal to the operation center, that gurantees a live time assistance to the driver.

2.2 Features analysis

We start our work by analyzing the observations collected during February 2018. For each alarm the following information are given:

1. *kpi_1*: key performance indicators(from *kpi_1_1* to *kpi_1_21*) referring to the accelerometer; they are all continuous variables

2. *check_2*: dummy variables(from *check_2_1* to *check_2_6*) referring to the accelerometer

3. *check_3*: as for check_2 they are KPIs(from *check_3_1* to *check_2_?*) referring to the accelerometer, they are all binary features

4. *kpi_4*: KPIs(from *kpi_4_1* to *kpi_4_8*) referring to the car position;

5. *history_5*: five KPIs relative to historical data of the black boxes, they are both continuous, both discrete variables

6. *check_ 6*: dummy features referring to the context and the quality of the data sent by the black box

7. *feedback_ assistenza*: this variable tells us if the signal corresponds to a real accident or if it is just a false crash, in fact it can assume only 5 different values:

   - 0: if the observation is a false crash and the driver did not receive any call or message
   - 1: if the feedback is missing, so we do not know whether it is a real or a false alarm
   - 2: when the driver received a call, but he did not have an accident
   - 3: if the driver has been called by the assurance and he really needed assistance
   - 4: when the driver received a call and he needed a tow truck

8. *chiamata_ autonoma*: this KPIs is 1 when a driver had an accident and needed assistance, but he did not receive any call

9. *identifier of the black box* KPIs which describes a black box, indicating its hardware and the type of installation

10. *identifier of the accident*: some attributes that identify an event :

    a *id_ evento*: code of the accident,

    b *tipo_ evento*: indicate if in the event is involved a car or a motorbike,

    c *istante_ evento*: date and hour of the accident

11. *identifier of the black box*: some attributes that identify a box :

    a *id_ blackbox*: code of the box,

    b *tipo_ installazione*: categorical variable, which indicates how the black box is installed

    c *classe_ hardware*: categorical variable, which shows the hardware of the black box.

Since our goal is to predict if the signal received is a false alarm or real crash, we need a criterion in order to split our data into these two groups. We use a criterion based on the value of the attributes *chiamata_ autonoma* and *feedback_ assistenza*: if *chiamata_ autonoma* = 1 the observation refers to a accident while, obviously, *chiamata_ autonoma* = 0 the signal refers to a false allarm. On the other hand, when the latter is equal to 3 or 4 the observation is classified as an accident, while when its value is 0 or 2 the signal is considered as a false crash. Finally we discard the data with *feedback_ assistenza* = 1 because we do not have any information about the customers' feedback.

Starting our analysis we find out that in our data set there are 4 attributes presenting missing values: *kpi_ 1_ 13* (227 values missing), *kpi_ 1_ 14* (37 values missing), *kpi_ 1_ 15* (602828 values missing), *kpi_ 4_ 6* (411695 values missing) and so we decide not to do take into account the last two attributes since the number instances for which this value is not available is considerable.

2.3 Preliminary analysis

From the insurance company that provided us the data, we know that there could be some alarms with a wrong timestamp or that there could be some corrupted black boxes that send continuously false crash signals. So we decide to check if the two phenomena are correlated with the black boxes class hardware or type of installation. To analyze the former scenario, since we have only the alarms from February we extract the samples presenting surely a wrong timestamp: the ones antecedent the midnight 1st of February and the ones after the 23:59 28th of the same month. On the other hand, to check the second situation we count for each black box how many alarms were sent by it and we find out some corrupted black boxes that send more than one hundred signals per month. We finally check out if wrong signals are related to black boxes properties, but there are no correlation between these variables, so we decide not to do take further action on this samples.

After dividing our features based on the type of KPIs, we start analyzing their distributions just calculating some quantity as means, standard devia-

tion, quantiles, etc for the continuous attributes, and we report these values in the following tables(Tables 2.1, 2.2, 2.6).

	kpi_4_1	kpi_4_2	kpi_4_3	kpi_4_4	kpi_4_5	kpi_4_6	kpi_4_7	kpi_4_8
mean	13.32	13.20	-1.87	0.00	37.91	57.07	6045.80	4081.19
std	23.62	23.62	23.21	0.87	42.25	183.63	51905.94	51673.64
min	0.0	0.0	-2987.68	-12.14	0.0	-1.0	0.0	0.0
25%	0.0	0.0	-0.10	0.0	8.0	-1.0	782.34	331.84
50%	0.0	0.0	-1.85e-05	0.0	14.0	0.0	1876.58	745.34
75%	21.2	20.21	0.0	0.0	69.0	14.42	4335.01	1871.03
max	173.84	192.0	65.99	22.01	333.0	8607.02	3966266.43	3961302.75

Table 2.1: Description kpi_4

	history_5_1	history_5_2	history_5_3	history_5_4	history_5_5	history_5_6
mean	0.02	1074.76	1055.20	161.46	2.94	3.88
std	0.18	2660.89	2642.48	843.23	3.61	4.17
min	0.0	0.0	0.0	0.0	0.0	0.0
25%	0.0	0.0	0.0	0.0	0.0	0.0
50%	0.0	61.0	55.0	5.0	1.24	3.45
75%	0.0	881.0	844.0	73.0	5.037	6.30
max	6.0	18962.0	18961.0	11573.0	22.84	36.46

Table 2.2: Description history_5

From these tables, we can notice that KPIs can assume very different values even looking among indicators derived by the same components. For instances, kpi_4_3 goes from -2987.68 to 65.99, while kpi_4_7 assumes values from 0 to 3.96e+6. For that reasons rescaling our data will be very important before starts our analyzes, in fact most part of machine learning algorithms are scale sensitive. Moreover, looking at the large differences between the third quartile values and the maximum, or between the minimum and the first quartile, we suppose that our data set could present outliers, that could affect our estimations. On the other hand, for the discrete or dummy variables, we simply write down the frequency of each class for each attribute.

From Table 2.3 we can notice that, as supposed, there are few samples with variable chiamata_autonoma equal to 1, which correspond to a serious

	chiamata_autonoma	tipo_evento	feedback_assistenza
class 0	99.93%	98.80%	98.80%
class 1	0.07%	1.20%	1.20%

Table 2.3: Description dummy variables

	check_2_1	check_2_2	check_2_3	check_2_4	check_3_1	check_3_2
class 0	99.48%	78.02%	90.94%	71.03%	99.95%	93.32%
class 1	0.52%	21.98%	9.06%	28.07%	0.05%	6.68%

	check_6_1	check_6_2	check_6_3	check_6_4	check_6_5	check_6_6
class 0	1.08%	0.67%	0%	0.03%	1.73%	0%
class 1	98.92%	99.33%	100%	99.97%	98.27%	100%

Table 2.4: Description dummy variables

	feedback_assistenza	classe_hardware	tipo_installazione
class 0	97.51%	7.42%	0.67%
class 1	0.35%	5.22%	0.3%
class 2	1.27%	3.93%	12.53%
class 3	0.67%	2.52%	38.65%
class 4	0.20%	0.42%	0.0005%
class 5	-	49.60%	1.63%
class 6	-	5.42%	0.005%
class 7	-	11.79%	45.94%
class 8	-	1.59%	0.51%
class 9	-	12.10%	0.04%

Table 2.5: Description dummy variables

accident did not recognized. Furthermore, we understand that the most part of the observations correspond to a false crash. Looking at Table 2.4, we choose to drop check_2_1 and check_3_1, because they are equal to zero in the most part of the samples. Moreover we decide not to use the dummy variable describing the context and the quality of data(*check_6*), because they assume all the value 1 in at least the 99% of observations.

	kpi_1_1	kpi_1_2	kpi_1_3	kpi_1_4	kpi_1_5	kpi_1_6	kpi_1_7	kpi_1_8	kpi_1_9	kpi_1_10
mean	3.69	18.69	8.02	1.50	1.54	1.70	0.13	1.85	746.10	0.19
std	4.40	102.01	27.39	7.99	8.03	8.57	0.26	4.72	815.43	0.32
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
25%	1.10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	98.0	0.0
50%	1.98	0.0	0.0	0.0	0.0	0.0	0.0	0.0	580.5	0.0
75%	4.31	4.70	5.0	0.13	0.13	0.25	0.08	1.0	1162.0	0.36
max	46.18	2378.20	492.0	204.29	448.80	441.67	1.0	133.0	4849.0	0.99

	kpi_1_11	kpi_1_12	kpi_1_13	kpi_1_14	kpi_1_15	kpi_1_16	kpi_1_17	kpi_1_18	kpi_1_19	kpi_1_20
mean	0.18	0.14	1.32	1.46	133.84	654.39	1.85	17.17	21.41	2.41e+16
std	0.31	0.28	7.38	7.86	219.03	184.73	4.03	25.43	28.20	3.92e+17
min	0.0	0.0	-1.0	-1.0	-1.0	0.0	0.0	0.0	0.0	0.0
25%	0.0	0.0	0.0	0.0	-1.0	592.0	0.0	1.44	2.02	8.98
50%	0.0	0.0	0.0	0.0	-1.0	592.0	0.0	4.23	6.15	24.50
75%	0.35	0.0	0.12	0.12	317.0	804.0	1.93	19.91	32.87	189.48
max	0.99	0.99	203.97	448.96	5442.0	6122.0	44.64	90.0	90.0	2.40e+19

Table 2.6: Description kpi_1

Chapter 3

Methodology

In this chapter we will explain all the steps that we follow in order to create a statistical learning model able to predict the nature of a signal. After further analysis on our data, we will start applying some unsupervised learning techniques, then we will create two training sets and we will use some supervised methods to fit different models. Finally, we will define the criterion that we will use tho choose the best classifier among all, we will test models, and find the most accurate one.

3.1 Exploratory analysis

First of all, since many machine learning estimators require the standardization of the data set to have good performances, we choose to normalize the continuous variables, using the implemented function available on Scikit-learn package *RobustScaler*. This scaler instead of removing the mean and scale the variance to unit, considers the Interquartile Range(IQR) in order to reduce the influence of the outliers. The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile). Thanks to Robustscaler function, better results respect to the usually normalization are obtained.

Furthermore we check if the features in our data set are significantly correlated, calculating for each couple of variable the Pearson's correlation coefficient. It is obtained by dividing the covariance of the two variables by

the product of their standard deviations. The correlation coefficient $\rho_{X,Y}$ between two random variables X and Y with expected values μ_X and μ_Y and standard deviations σ_X and σ_X is defined as

$$\rho_{X,Y} = \text{corr}(X,Y) = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

where E is the expected value operator, *cov* means covariance, and *corr* is the notation used for the correlation coefficient. The correlation coefficient can assume values between -1 and 1; it is +1 in the case of a perfect direct linear relationship, while it is -1 in the case of a perfect inverse linear relationship. The closer the correlation is to zero the more uncorrelated are the variables.

After that we analyze the distribution of all the continuous KPIs, starting from some basic plots as the one represented in Figure 3.1. The plots of the left columns are made using all the observations, while the plots of the right columns take into account only the observations with a feedback by the driver. The plots of the first row represent the distribution of the samples, the second row contains boxplots and finally the third row shows again boxplots, but this time dropping the outliers.

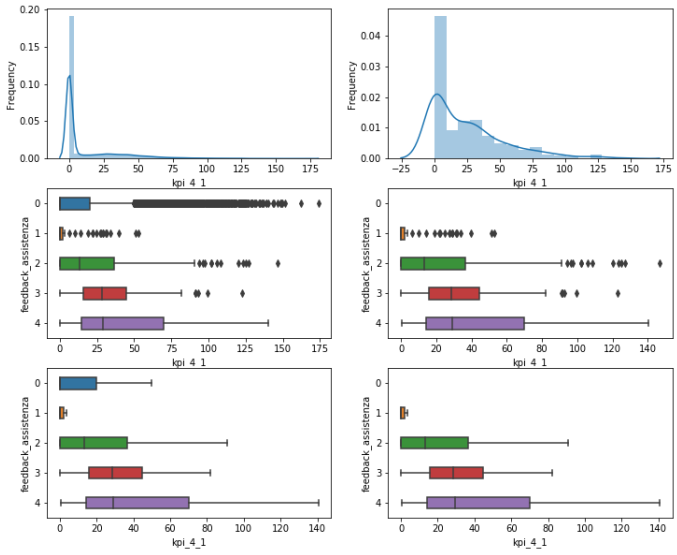


Figure 3.1: Distribution analysis of kpi_4_1

In order to detect outliers more precisely, we would like to fit a multivariate distribution of the KPIs referring to accelerometer, position and historical information of the black boxes. First of all we try to fit the distribution of continuous KPIs one by one to check if the results obtained for univariate estimation are accurate, before proceeding with the multivariate one. The procedure that we follow is very easy: first we fit some possible distributions, such as normal, exponential, lognormal, etc. using the *fit* function of the package *Scipy* and then we find which is the best one through the Kolmogorov-Smirnov test, already implemented in the package *Scipy*. After that, in order to compare the distribution found with the real one, some random samples are drawn and then compared through a histogram to the original data.

We decide not to use multivariate estimation to detect outliers and we choose another technique : the isolation forests. The idea on which isolation forests are based is that, when we train a random forest on a data set, the number of splittings required to isolate a sample in a tree can be seen as a measure of the abnormality of an observation, the lower is this value the higher is the probability of facing an outlier.

3.2 Unsupervised learning

3.2.1 Cluster analysis

The goal of clustering is to identify groups in data, in order to compress or to segment data; for example a company that employs K sales persons wants to split the customer database into K parts each comprising customers as similar as possible. Sometimes the number of clusters K is fixed before looking at the data , while often K is unknown and so we would like to find the best value of K to understand at best our data. The main problem is that increasing K will typically decrease within-cluster sum of squares $W(C)$, where

$$W(C) = \sum_{k=1}^{K} \sum_{C(i)=k} \|x_i - c_k\|^2 \tag{3.1}$$

and $c_k = N_k^{-1} \sum_{C(i)=k} x_i$, so we can not use the minimization of this value to find the best number of clusters. The most common method to choose K is based on the following heuristic: if the data truly consist of K^* clusters and we use clustering algorithms with increasing values of $K < K^*$, we expect $W(C)$ to decrease substantially when true clusters will be identified, however for $K > K^*$ we expect the decrease in $W(C)$ to be small. Then an 'Elbow approach' is used, we make a plot of $W_K(C)$ for increasing values of $K = 1, 2, ...$ and try to locate a 'kink' in the curve.

This procedure works poorly when the correct number of cluster is $K^* = 1$ and it is not always clear which K is the best one. Tibshirani et al. (2001) introduced the gap statistic which is based on the same heuristic as the elbow method, but gives better results especially for $K^* = 1$. The gap statistic is based on a randomization idea: compare the curve of $logW_K(C)$ obtained from the data $\{x_i\}_{i=1}^n$ to the curve obtained from data uniformly distributed (no clustering); in formula Gap statistic:

$$G(K) = E(logW_K^{indep}) - logW_K$$

where the expectation is under the random distribution. The expectation is evaluated by (Monte Carlo) simulation: for $b = 1, ..., B$, generate n uniformly distributed points over a rectangle containing the $\{x_i\}_{i=1}^n$ and compute the corresponding curve $logW_K^{(b)}$. Then

$$E(logW_K^{indep}) \approx \frac{1}{B} \sum_b logW_K^{(b)}$$

defining K^* as the value of K where the gap between the two curves is largest.

K-means

First of all, we decide to use k-means algorithms in order to divide our data, because we know that we want only two clusters and, most of all, it is one of the least expensive in term of computational cost. Assume that we want to divide our original data in K clusters, where K is fixed, and small compared to n, the number of observations, K-means algorithms estimates a function, called encoder, $C : 1, ..., n \to 1, ..., K$ which maps each x_i to a cluster index $k \in 1, ..., K$. The goal of K-means is to find C to minimize the within-cluster

sum of squares(Equation 3.1). In general, it is impossible to optimize $W(C)$ over all possible partitions, so we have to use an iterative algorithm to find a local minimum of $W(C)$. The steps of K-means algorithm are

1. Given a cluster assignment C, calculate the cluster means $c_1, ..., c_K$.

2. Given $c_1, ..., c_K$, update C such that each x_i is assigned to the cluster k corresponding to the closest center, i.e.,

$$C(i) = argmin_{k=1,...,K} \|x_i - c_k\|^2.$$

The optimal solution of K-means is

$$C^* = argmin_C W(C) = argmin_C \sum_k \sum_{C(i)=k} \|x_i - c_k\|^2 = argmin_{C,m_1,...,m_K} \sum_k \sum_{C(i)=k} \|x_i - m_k\|^2.$$

Clearly K-means algorithm converges to different local minima, depending on which initial values are chosen as cluster centers, so we have to run K-means algorithm several times with different initial values, and keep the encoder corresponding to the smallest $W(C)$. Since the KPIs composing our data set represent completely different aspects of the event that we are analyzing, we decide to apply k-means to different subset of indicators. Therefore three groups of variables are created, one using accelerometer KPIs, another one using positions information and, finally, one considering only the historical attribute indicators.

In order to control the validity of our clusters we compare for each division calculated the value of the *adjusted rand index score*. This score compare the labels assigned by the cluster algorithm and the real target values of the observations, it can assume values in the interval $[-1, 1]$, where 0 indicates that no one of the labels has been predicted correctly, while if the adjusted rand index score is equal to 1 that means that the division created by the cluster is equivalent to the original one. Moreover the adjusted rand index score is not influenced by permutation of the predicted labels. For example, supposing we have a data set with 4 samples where target variable assume values $[0, 1, 0, 1]$ and the predicted labels are $[0, 1, 0, 1]$ in one case and $[1, 0, 1, 0]$ in another one, in both the situations ARI will be equal to

one, since there is perfect correspondence between the real division of the data and the predicted one.

Moreover, in order to plot the results of k-means, we project our data into a 3-D space using principal component analysis(PCA). To summarize, the pipeline used in this phase is:

- normalize our data using RobustScaler
- apply k-means algorithm, fixing the number of clusters equal to 2
- project our data using PCA
- calculate adjusted rand index
- plot results and original labels

DBSCAN

In order to mitigate the effects of the outliers, we decide to use DBSCAN algorithm. DBSCAN could be useful to detect and eliminate extreme samples, but on the other hand, we can not decide a priori the number of clusters and it is computationally more expensive than k-means. The DBSCAN algorithm finds clusters separeting areas of high density by areas of low density; so clusters can be any shape, while using k-means the groups found are convex shaped. The main idea of DBSCAN is that a cluster is composed by a set of core samples, each close to each other (measured by some distance) and a set of non-core samples that are close to a core sample. An observation is called a *core sample* when there are at least min_samples other samples within a distance of eps, if this happens the sample is in an area of high density. A cluster has a set of non-core samples too, which are observations that are neighbors of a core sample in the cluster but are not themselves core samples. A cluster can be built by recursively taking a core sample, finding all of its neighbors that are core samples, finding all of their neighbors that are core samples, and so on. Any observation that is neither a core sample neither a neighbor of a core sample is considered an outlier by the algorithm, since it is not contained in any cluster. As explained before, DBSCAN has two parameters that need to be tuned: the number of neighbours for each

center node, and a distance *eps*. In order to choose the best values for the parameters above, we try different numbers and we choose the couple that maximize the value of ARI.

3.2.2 Manifold learning

In data analysis one of the main problems in preprocessing data before applying any supervised or unsupervised algorithm is the features extraction. In fact the presence of correlation in our data set could lead to less precision or less predictive power in the model found. Analyzing an high-dimensional feature space could slow down a lot the algorithm used. That is why in the phase of preprocessing, dimensionality reduction is required. The most common method is principal component analysis(PCA). Thanks to linear transformation, PCA projects a data set onto a lower-dimensional space. It is usually used for visualize our samples, in order to recognize some particular pattern, not visible through human eye in high-dimensional data set. One of the main drawbacks of PCA is that usually it can not identify non linear relation in our data. Therefore in this work we decide to use another technique called manifold learning, that allows us to project our data on a lower-dimensional space, using non-linear transformation.

We try three nonprobabilistic methods to perform manifold learning: multidimensional scaling(MDS), locally linearly embedding(LLE) and isometric feature mapping(isomap). All these algorithm are based on the definition of a similarity matrix between the samples and, then, the observations are projected into a space with a lower defined dimension, trying to preserve the original similarities between points.

Multidimensional scaling

Multidensional scaling (MDS) algorithm starts from defining a distance matrix that contains the pairwise distance between any possible couple of sample. After that the goal of MDS is to find a low-dimensional projection of the data, calculating and using the eigenvector of the distance matrix, that preserve, as closely as possible the distances between data points. The distance between point can be calculating both using the Euclidian norm,

both using some similarity or dissimilarity measures, giving origin to non-metric MDS.

Locally linear embedding

Locally linear embedding, or LLE needs the definition of a neighbourhood for each sample x, so choosing a number n, the n nearest points will be considered neighbours of x. After that, the local geometrical properties of the neighbourhood are defined, computing for each sample a set of coefficients with particular property. In particular, they have to be invariant to linear transformation such as rotation, scaling and translation. Then data points are mapped from the original feature space to a lower dimensional one, keeping the coefficients calculated at the previous step constant. Since the projection of the samples from a one-dimensional space to another one is done using linear transformation and the weights are defined to be invariant to those, the local geometry and the coefficient calculated will be preserved in the new data set, too.

Isometric feature mapping

In isometric feature mapping, or isomap, we use the same technique of MDS, but this time we define the distances matrix using the geodesic distance. So, as for LLE, we have to define a neighborhood that can be established selecting the K nearest neighbours or including all the points whose distance from the sample in analysis is less then a hyperparameter ϵ. The algorithm then draws a graph where each sample is connected by an edge with its neighbours and assign to the edge a weight equal to the distance of its vertices. Then as explained in 'Pattern Recognition and Machine Learning'(Bishop [6]) "the geodesic distance between any pair of points is approximated by the sum of the arc lengths along the shortest path connecting them". Finally, the algorithm applied the same procedure of metric MDS to the distances matrix, finding the projection onto a new low-dimensional space.

Tuning hyperparameters

Each of these algorithms has some hyperparameters that need to be tuned. In order to do so, we calculate the projections assigning various values to the hyperparameters and then we compare the stress indicator for MDS and the reconstruction error for LLE and isomap. The values that have to be tuned for each method are:

- MDS: n_components, number of dimensions in which to immerse the dissimilarities and metric, that indicates whether to perform metric MDS; or nonmetric MDS.
- LLE: n_neighbors , number of neighbors to consider for each point and n_components, number of coordinates for the manifold
- isomap: n_neighbors , number of neighbors to consider for each point and n_components, number of coordinates for the manifold

3.3 Supervised learning

In this section we will describe how we derived classification algorithms to decide whether a signal identifies a real accident or a false crash. Most classifiers perform badly when used with unbalanced data set, like the one we are analyzing. This is because most models minimize the misclassification error which is a symmetric loss. Hence such models will focus more on the prediction accuracy of the most common class and will often give poor results for the other classes. We try two different approaches to solve problems caused by the the fact that our data set is unbalanced: the first is using a smaller but balanced training set, the second one is to consider a larger dat set, but assigning to each observation a different weight, based on which class it belongs.

Therefore we create two different training sets: the former contains few observations($\sim$ 19000) which are almost equally distributed between false alarms and real crashes, while the latter contains more signals($\sim$ 550000) but where the number of real and false crashes is not balanced. All the other observations are used for testing our models, so in the first test set about

900000 observations are contained, while in the second set there are only about 450000 observations.
In the previous section we explain that, we try to do features selection using some multidimensional scaling techniques, but in practice without obtaining relevant results. So, in order to verify if all the KPIs available are significant for our previsions we decide to split the our data set into smaller ones. Table 3.1 shows which KPIs compose each subset used on our analysis.

Data set	*KPIs*
kpi_1	accelerometer KPIs
kpi_4	position KPIs
kpi	accelerometer and position KPIs
acce	accelerometer KPIs and check
kpi_1c	accelerometer KPIs, identifier of the accident and of the black box
kpi_4c	position KPIs, identifier of the accident and of the black box
kpic	accelerometer and position KPIs, identifier of
	the accident and of the black boxes kpic and the check accelerometer
kpih	black boxes historical data and accelerometer and position KPI
allh	same KPIs of *kpih* plus accelerometer check
allhc	all KPIs avaiable
allhc	all KPIs avaiable plus three dummy variables that indicate
	the time slot of the day when the accident occurs

Table 3.1: Subset of KPIs used for each data set

Data set	**Training set 1**	**Test set 1**	**Training set 2**	**Test set 2**
Observations	550689	373481	17449	906721
False alarms	544122	371271	10882	904511
Crash	6567	2210	6567	2210

Table 3.2: Training and test sets used

3.3.1 Algorithms used

This section summarizes the mathematical derivations, advantages, and disadvantages of each algorithm used in the supervised phase.

Decision tree classifier

Decision trees stratify the feature space recursively into simple regions. Even if trees are simple and easy to interpret, they are usually not very good at prediction, nevertheless modern methods based on trees (random forests, boosted trees) perform better. A decision tree can be represented as a graph, where each split is called node, terminal node is called a leaf and interior nodes lead to branches. In order to divide the feature space into M non-overlapping regions $R_1, ..., R_M$ a top-down, greedy approach known as recursive binary splitting is used: begin at the top of the tree with a unique region and divide the feature space into two regions, and then divide these subregions into two, etc. . . The prediction $\hat{y}_+$ for a new x_+ falling into some region R_m is given by

- Classification: the majority class into R_M;
- Regression: the average of the y_i for which the corresponding $x_i \in R_M$.

Trees are models of the form:

$$T(x) = \sum_{m=1}^{M} c_m I(x \in R_m) \tag{3.2}$$

The $c_m \in \mathbb{R}$ are estimated from the observations (x_i, y_i) with $x_i \in R_m$. Now we analyze how to grow the tree and how to choose the splits. We first look at the case of regression with the squared error loss. We use a greedy approach based on binary splits to estimate f: starting at the top, for each coordinate $j \in 1, ..., p$ we look for the best binary split defining

$$R_1(j, s) = x \in \mathbb{R}^p : x_j \leq s \text{ and } R_2(j, s) = x \in R_p : x_j > s.$$

The values of $j \in 1, ..., p$ and $s \in \mathbb{R}$ are found by solving:

$$\min_{j,s} \left\{ \min_{c_1} \sum_{i:x_i \in R_i(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{i:x_i \in R_i(j,s)} (y_i - c_2)^2 \right\}$$

Note that $\sum_{i:x_i \in R_i(j,s)} (y_i - c_1)^2$ is minimized for $\hat{c}_1 = ave\{y_i : x_i \in R_i(j,s)\}$ and similarly for the second sum after finding the first split, the procedure is repeated in each branch.
In order to grow a classification tree we need an appropriate splitting criteria; for regression the squared-error node impurity measure is used, while for classification different loss functions can be used. Considering a region R_m with N_m observations we define the portion of a class k as

$$\frac{1}{N_m} \sum_{i:x_i \in R_m} I(y_i = k)$$

Some of the impurity measures most used are:

- *misclassification error*: $\frac{1}{N_m} \sum_{i:x_i \in R_m} I(y_i \neq k) = 1 - p_{mk}$
- *Gini index*: $\sum_{k \neq k'} p_{mk} p_{mk'} = \sum_{k=1}^{K} p_{mk}(1 - p_{mk})$
- *Cross-entropy (or deviance)*: $-\sum_{k=1}^{K} p_{mk} log(p_{mk})$

In the case of two classes, the misclassification error is $1 - \max(p, 1-p)$, the Gini index is $2p(1-p)$, and the cross-entropy is $-plogp - (1-p)log(1-p)$. The misclassification impurity is non-differentiable, so it is not good for optimization while the Gini index and the cross-entropy will favor pure nodes with $p_{mk} \approx 0$ or $p_{mk} \approx 1$. In our analysis we use the Gini index.

As explained before, decision-tree learners can be affected by overfitting, so in order to avoid this problem, the minimum number of samples required at a leaf node or the maximum depth of the tree should be tuned. Moreover decision trees can be unstable because small variations in the data can generate a completely different tree and the model created could be biased trees if some classes dominate. For this reason is better to weight the samples before train a decision tree.

The main disadvantages of the decision trees could be mitigated by training multiple trees in an ensemble learner. The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm in order to improve robustness over a single estimator. Two families of ensemble methods are usually distinguished:

- averaging methods, such as random forests.

- boosting methods.

The main idea on which averaging methods are based is to build several trees independently and then to average their predictions. On average, the combined model has more predictive power and it is less influenced by the presence of outliers, since the variance of the model is reduced by averaging. In next sections, the ensemble methods used in this work are explained.

Random forest classifier

Random forest is an ensemble method based on averaging tree, where the trees in the forest are constrained in order to decrease their correlation. The algorithm consists in bootstrap the data B times, then when growing each tree on the bootstrap samples, for each split select m of the possible p variables to be used for the split where this subset of variables changes for each split. If we want to predict the variable $\hat{y}_+$ for a new x_+ in case of regression we use the average the predictions from the B trees, while in case of classification we select the majority vote from the B trees. When one variable is much more important than the others then all bagged trees will select this variable for the first split, making these trees similar, hence correlated, in order to avoid this we select a random subset of m variables for each split. In practice the value $m = \sqrt{p}$ for classification and $m = p/3$ for regression works well in practice. Even if we loss the interpretability of the single tree, random forests have great predictive performance and need very few tuning parameter. It can be shown that on average each bootstrap sample contains approximately 63% of the data (because of the sampling with replacement). This means, for each tree in the forest, approximately 37% of the data are not used for the fit, so we can use these data to estimate the prediction error from this tree, and for each observation (x_i, y_i) we can consider all trees not containing this observation and estimate a prediction error. Doing this for all observations and for all trees in the forest finding the out-of-bag (OOB) error estimate, which is equivalent to the cross-validation error. Moreover there are different measures of variables importance in random forests:

- *mean decrease variable importance*

- *permutation variable importance*

The former could be biased, so it is better to use the latter: to calculate the importance of the jth variable, X_j we consider an error estimate from the OOB, and we permute its values (randomly) for the OOB observations and then measure the increase in prediction error; the more the error will increase the more important is the variable.

Boosting classifiers

The main idea of boosting is slow learning: combine weak learners to slowly learn a regression function or a classifier. A weak learner is a model that performs only slightly better than random guessing, as small trees, leading to so-called gradient tree boosting. There are many different boosting algorithms depending on the choice of the weak learners used and the way they are combined. The idea of combining weak learners to improve their performance appeared in the 90s in computational learning theory literature (Schapire and Freund). First boosting algorithm was AdaBoost (Freund & Schapire, 1996), while XGBoost (Chen & Guestrin, 2016) makes boosting one of the most efficient predictive model. We now analyze in details the algorithm used in this work.

ADABoost classifier We consider the AdaBoost.M1 algorithm (Friedman et al., 2000). Suppose we want to find a classifier using training data $\{(x_i, y_i)\}_{i=1}^{n}, y_i \in \{-1, +1\}$. The steps of AdaBoost.M1 (ESL, algorithm 10.1):

1. Initialize the weights $wi = 1/n, \quad i = 1, ..., n.$
2. For $m = 1, ..., M$
   - **(a)** Fit a weak classifier $G_m(x)$ to the training data using weights w_i.
   - **(b)** Compute
   $$err_m = \frac{\sum_{i=1}^{n} w_i I\{y_i \neq G_m(x_i)\}}{\sum_{i=1}^{n} w_i}$$
   - **(c)** Compute $\alpha_m = log(1 - err_m)/err_m$
   - **(d)** Set $w_i \leftarrow w_i \times exp\left[\alpha_m I\left\{y_i \neq G_m(x_i)\right\}\right], i = 1, ..., n$

3. Output $G(x) = sign\left\{\sum_{m=1}^{M} \alpha_m G_m(x)\right\}$

Note that the final decision is based on a linear combination of the weak learners G_m and that each classifier $G_m(x)$ returns a prediction of ± 1. If we want to classify weighted data we have to use a loss function which is a weighted sum of the form $\sum_{i=1}^{n} w_i L(y_i, \hat{y_i})$ The error for each classifier will be between 0 and 0.5, while the term α_m is 0 when $err_m = 0.5$ and tends to $+\inf$ when $err_m \to 0.5$. Larger α_m means that the sign of G will be very influenced by the corresponding G_m.

Gradient Boosting classifier Friedman et al. (2000) proposed gradient boosting machine (GBM), which uses the following functional gradient descent algorithm to estimate f^*: starts with the null function $f(0)(x) = 0$, and for $t = 1, ..., T$ defines

$$f(t+1) = f(t) + \gamma G(t+1),$$

where the function $G(t+1)$ is a weak learner fitted to the sample

$$\{(x_i, \hat{y}_i)\}_{i=1}^{n} \text{ where } \hat{y}_i = -\frac{\partial err_f}{\partial f_i}(f^{(t)})$$

At each iteration of the algorithm we use a weak learner G to approximate the gradient of the function err_f, in particular Gradient tree boosting uses small trees for G: the size M of the trees is fixed.

At each iteration t we fit a regression tree $G(t+1)(x) = T(x;\theta)$ to the sample

$$\{(x_i, \hat{y}_i)\}_{i=1}^{n} \text{ where } \hat{y}_i = -\frac{\partial err_f}{\partial f_i}(f^{(t)})$$

The vector θ contains the tree parameters: the regions $R_1, ..., R_M$ and their corresponding predicted values $c_1, ..., c_M$. To estimate the "best" θ a two-step approach :

1. The structure of the tree, i.e., the regions $R_1, ..., R_M$ are found by fitting a regression tree of size M to $\{(x_i, \hat{y}_i)\}_{i=1}^{n}$

2. Given the regions $R_1, ..., R_M$, the values of $c_1, ..., c_M$ are found by minimizing, for each $m = 1, ..., M$,

$$\sum_{i=1}^{n} I(x_i \in R_m) L(y_i, f^{(t)} + c_m)$$

The final estimate $\hat{f} = f^{(T)}$ is an additive model in the adaptive basis functions G, i.e., $\hat{f}$ is a linear combination of the weak learners $G^{(T)}$:

$$\hat{f} = \sum_{t=1}^{T} \gamma G^{(T)}(x)$$

At each iteration t, GBM fits a small tree to the current residuals, and then add this tree to the current model with a small weight γ, we use small trees to avoid fitting models that are too complex and we multiply tree by the value γin order to learn slowly. The number of iterations T and the shrinkage parameter γ impact the test error of boosting: when γ is large the gradient descent algorithm can converge faster to the minimum, while when T is large boosting will be able to decrease the training error more. T and γ affect the performance of boosting jointly: for smaller γ we need larger T to obtain similar error, however it was found on examples that the best practice is to take γ small and T large, in fact boosting will overfit for very large T, but if γ is small overfitting will only appear for very large T. Another way to regularize boosting is to use a subset of the features similarly to random forests, finding OOB error and variable importance, too.

3.3.2 Neural networks

Finally we decide to consider models trained using Multi-layer Perceptron. MLP is a supervised learning algorithm that learns a function $f(\cdot) : R^m \rightarrow R^o$ by training on a data set, where m is the number of dimensions for input and o is the number of dimensions for output, so it can be used for regression problems or classification models, too. The advantages of Multi-layer Perceptron are the capability to learn non-linear models and the capability to learn models in real-time, keeping the model always updated to new scenarios. Otherwise MLP with hidden layers have a non-convex loss function where there exists more than one local minimum. Therefore different random weight initializations can lead to different validation accuracy.

3.3.3 Tuning parameters

The hyper-parameters of a model are parameters that are not directly learnt within estimators, such as the number of estimators in a random forest or the maximum depth of a tree.

When evaluating different settings for estimators, there is still the possibility to find a model that presents overfitting, because the hyperparameters can be tuned until the estimator performs optimally. This way, knowledge about the test set can lead to a classifier that predicts correctly the target value only in particular situations. To solve this problem, we can consider another subset of observations, creating the so called validation set: we fit our model on the training set, we check its performance on the validation set, and when the results are satisfying, we do our final evaluation on the test set. However, by partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model, and the final results can depend on a particular random choice for the pair of (train, validation) sets.

To avoid this problem usually is used a cross-validation(CV) approach. A test set should still be taken into account out for final evaluation, but, in case of CV, we do not need anymore the validation one. In the k-fold CV, the training set is split into k smaller sets. For each partition a model is fitted using the remaining $k-1$ folds as training data, then the resulting model is validated on the remaining part of the data, calculating a certain score for the classifier. The final performance measure of our model is the average of all the scores calculating at the previous step. This approach is more expensive then dividing our original data set in train test and validation set, but allows us to include more samples in the training set and to find a model more robust. In this work we use the Sklearn function *GridSearchCV* in order to tune the hyperparameter of the classifier and, at the same time, to do cross-validation. We define a grid of possible values for the hyperparameter and the function *GridSearchCV* fits a model using CV for each possible combination of the parameters. For our models we tune the following hyperparameters:

- Decision Tree
  - *max_ depth*: maximum depth of the tree

  - *max_features*: maximum number of features used
  - *min_sample_leaf*: minimum number of observations for leaf

- Random forest
  - same parameters of decision tree
  - *n_estimators*: number of estimators composing the forest

- AdaBoost classifier
  - *n_estimators*: number of estimators composing the classifier
  - *learning_rate*: contribute of each classifier in the final model

- Gradient Boost classifier
  - *n_estimators*: number of estimators composing the classifier
  - *learning_rate*: contribute of each classifier in the final model
  - *max_depth*: maximum depth of each tree

- Neural networks
  - *hidden_layer_sizes*: number of neurons in each hidden layers
  - α: L^2 penalty (regularization term) parameter

During the validation phase, *GridSearchCV* needs a performance indicator to decide which combination of hyperparameters defines the best model. In our work we use the F_β, which is deeper described in the following section.

3.3.4 How to choose the best model

One of the request of the insurance company was to find different classifiers depending on the maximum number of signals that they could receive. Therefore we have to choose a performance indicator that could be easily adapted to new demands. We decide to use the F_β score calculated as:

$$F_\beta = \left(1 + \beta^2\right) \frac{precision \cdot recall}{\beta^2 \cdot precision + recall}$$

where

$$precision = \frac{number\ of\ crashes\ correctly\ classified}{total\ number\ of\ accidents}$$

$$recall = \frac{number\ of\ crashes\ correctly\ classified}{number\ of\ observations\ classified\ as\ an\ accident}$$

The main advantage of this indicator is that, just changing the value of β we can give more importance on precision or on recall. The parameter β can assume values in $(0, +\infty)$; when $\beta = 1$ the F_β considers at the same way precision and recall, on the other hand when $\beta \to 0$ F_β gives more importance to precision and, finally, if $\beta \to +\infty$ the indicator takes into account only the recall value. Looking only at precision means try to classify correctly the highest number possible of accidents, while paying more attention on the recall means try to predict the target value of more observations as possible in the right way.

3.4 Analysis tools

The analysis have been performed using the programming language Python[1] in the JupyterNotebook[2] interactive environment. Python allows a dynamic and efficient programming and relies on several packages and libraries that implements a wide number of functions in different fields. The following packages have been used throughout the code:

a *Matplotlib*[3] plotting library, used for most of the graphs and figures

b *NumPy*[4] package providing both data structures and functions for basic scientific computing

c *Pandas*[5]high-performance data structures and data analysis tools for large databases

d *Scikit-learn*[6] package providing tools for data mining, data analysis and machine learning

[1]https://www.python.org
[2]http://jupyter.org
[3]http://matplotlib.org
[4]http://www.numpy.org
[5]http://pandas.pydata.org
[6]http://scikit-learn.org

e *SciPy*[7] library of functions for scientific computing and statistical analysis

f *Seborn*[8] plotting library use for some of the graphics in this work. Computational resources provided by hpc@polito, which is a project of Academic Computing within the Department of Control and Computer Engineering at the Politecnico di Torino ()

[7]https://www.python.org
[8]https://seaborn.pydata.org/

Chapter 4

Exploratory analysis and unsupervised learning results

In this chapter and in the following one, the results obtained using method and algorithms explained in Chapter 3 are collected and illustrated. In particular in this chapter we will focus on the results of exploratory analysis and unsupervised phase.

We start calculating the correlation between any couple of KPIs, looking for particular linear relations. Moreover we use some plots in order to have an idea of how the attributes of the data set are distributed, trying to understand if they present outliers or if they assume different values when referring to false alarms or real crashes. After that, we try to fit the distribution of continuous variables, in order to check if it would be meaningful to look for a multivariate distribution, considering all the KPIs. If we do not succeed in fitting distributions, we will use isolation forest in order to detect the possible outliers present among the data. Furthermore, we apply different unsupervised techniques, in order to divide samples referring to false alarms from sample representing real crashes, using different clustering algorithms and applying them to different KPIs subsets. Finally, we use isomaps in order to reduce the number of features composing our data set and trying to decrease the high correlations present between the original attributes.

4.1 Exploratory analysis

As explained before, first of all we calculate the correlation between all the continuous KPIs present in our data set, dividing them by type. As shown

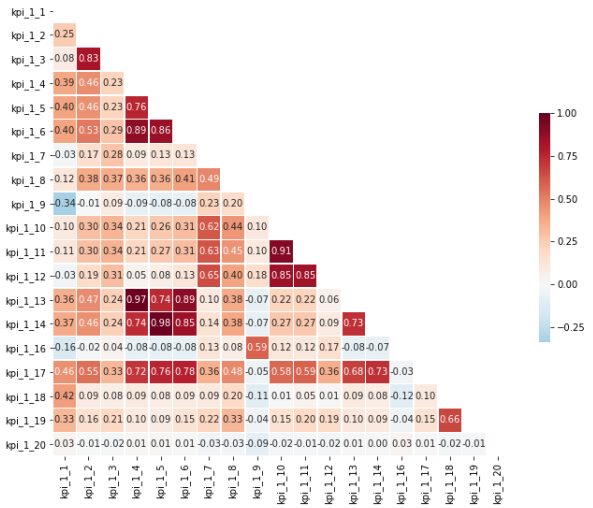


Figure 4.1: Correlation matrix kpi_1

in Figure 4.1, there are some groups of KPIs that have a very strong positive correlation as *kpi_1_10* and *kpi_1_11*, or *kpi_1_5*, *kpi_1_6*, *kpi_1_13* and *kpi_1_14*. Some of them, as the couple (*kpi_1_5*, *kpi_1_14*) show a correlation greater then 0.97. On the other hand, other KPIs referring to the accelerometer show smaller or negative correlation and *kpi_1_20* is the only one which is not correlated to any other indicator(for *kpi_1_20* all the correlation coefficients belong to the interval $(-0.1, 0.04)$). We now analyze the correlation coefficient for the other two subset of features, position and historical indicators. In Figure 4.2 we notice that there are only two couples of KPIs which are strongly correlated, *kpi_4_1* and *kpi_4_2*($\rho = 0.84$), *kpi_4_7* and *kpi_4_8*($\rho = 0.54$), while all the other pairs show small or null correlations. Finally in Figure 4.3, we notice that all the historical indicator , but *history_5_1* and *history_5_4*, are more or less correlated; the couple with the highest correlation coefficient is *history_5_2* and *history_5_3* with

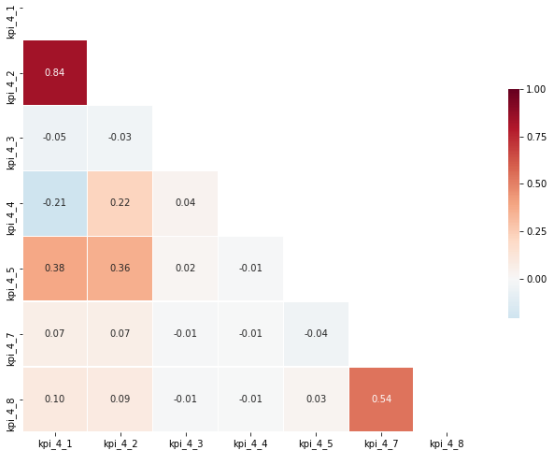


Figure 4.2: Correlation matrix kpi_4

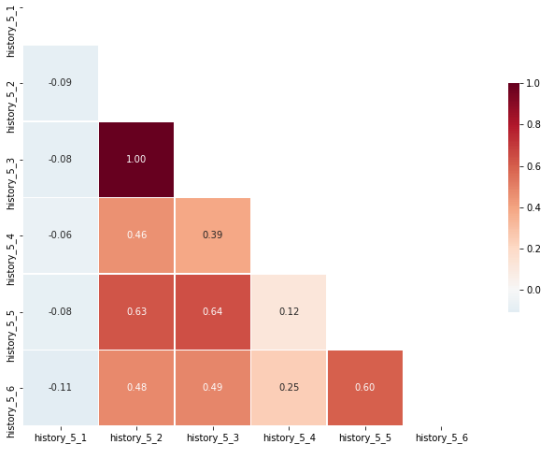


Figure 4.3: Correlation matrix history_5

a ρ equal to 0.99. After studying the correlations present in our data set, we try to understand more in depth the distribution of the features composing it. First of all we simply plot our data using histograms and box-plots,

trying to visualize our samples and their characteristics. From these plots, we understand that the data set in analysis presents lot of outliers and that there are some attributes(like *kpi_ 1_ 17* shown in Figure 4.4) which have different distributions if the samples refer to a false alarm or to a real crash. Since

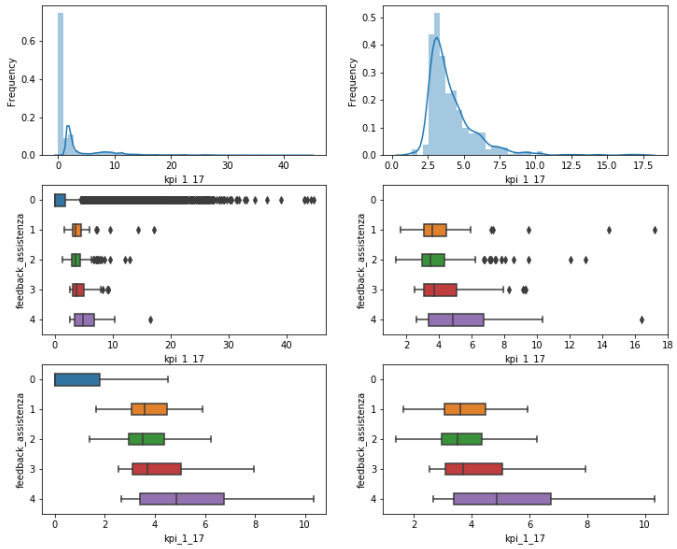


Figure 4.4: Distribution analysis of kpi_1_17

the presence of the outliers may affect the performance of the supervised methods, we identify and drop them. As explained in the previous chapter, we try to find the distribution of each KPI, but, as shown in Figure 4.5 the resulting distribution does not reflect the real one for the most part of KPIs, since they do not follow a conventional probability distribution. In Chapter 5 we will see how we use outlier detection in order to try to improve the predictive performance of the supervised classifiers.

4.2 Unsupervised learning

After understanding deeper the characteristics of the features composing our data set, we decide to apply some algorithms of unsupervised learning on it. In fact clustering could show us if the false alarms and the real crash

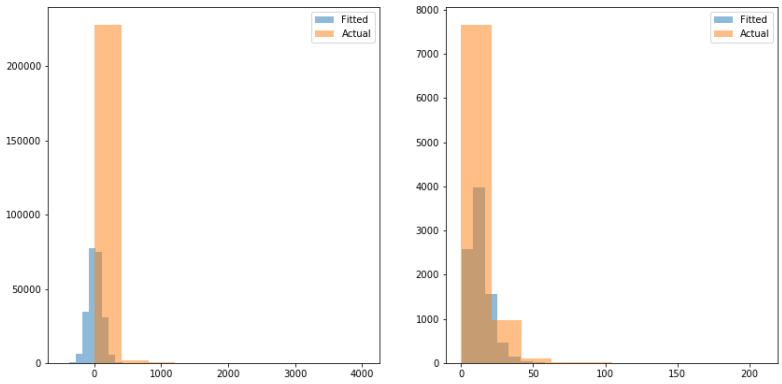


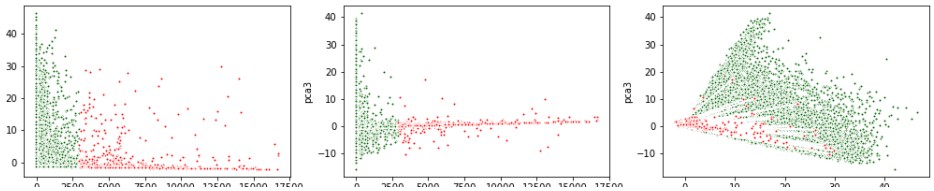
Figure 4.5: kpi_1_2 distribution and the simulated one

signals can be easily divided, or if there is some recognizable patterns within clusters.

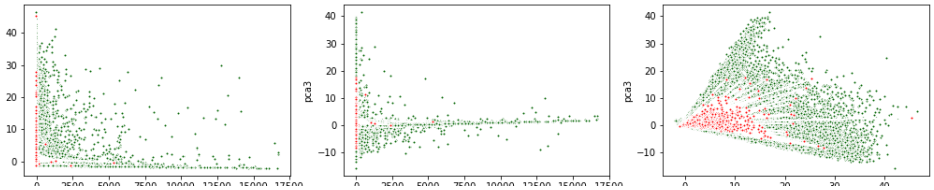
On the other hand, as denoted in the previous chapter, there are a lot of strong correlated variables and that our data set has a large number of attributes. The former property could affect the predictive power of some classifiers, derived in the supervised analysis, while the latter cause an increasing of algorithms computational cost. Therefore, in order to solve the possible problems evidenced above, some manifold learning techniques will be applied to our set of attributes. In the following sections and the result of cluster analysis and manifold learning will be explained.

4.2.1 Cluster analysis

First of all, we report the results of the applications of K-means pipeline implemented in the previous chapter. In the images below(4.6,4.7,4.8) there are the results respectively of the pipeline applied to *kpi_1*, *kpi_4*, *history_5*. Moreover from images we can notice that there is not a strong correspondence between real labels and predicted ones for any of the data set used, as evidenced by the value of the ARI, too. Another issue displayed by the Figure 4.7 is the presence of outliers that affect our estimations. As explained in the previous chapter, DBSCAN is computationally expensive, so we run this algorithm using only a subset of samples, containing about

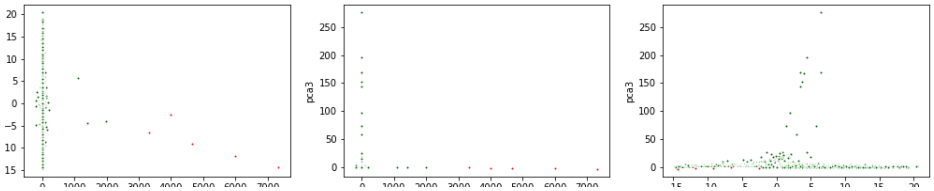


(a) K-means clusters using kpi_1 ARI=-0.0019

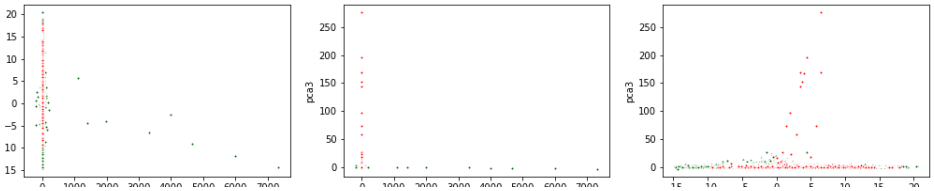


(b) Real label

Figure 4.6: K-means applied to kpi_1



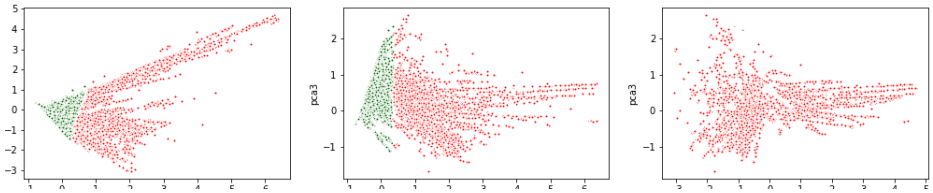
(a) K-means clusters using kpi_4 ARI=-1.34e-5



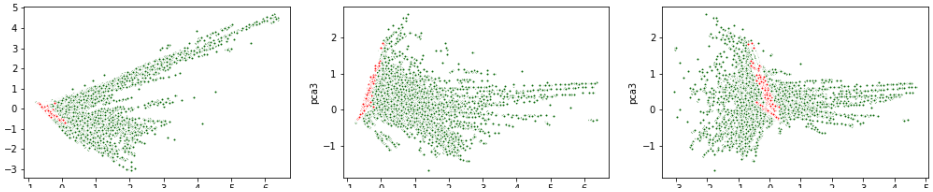
(b) Real label

Figure 4.7: K-means applied to kpi_4

18000 observations. In Figures 4.9, 4.10, and 4.11 are reported the clusters obtained through DBSCAN for KPIs of accelerometer, position and historical data. Most part of the observations have been marked as outliers, about 12000 when using kpi_1, 11000 with kpi_4 and 5000 from the applied on history_5 KPIs. Furthermore we can notice that no one of the divisions

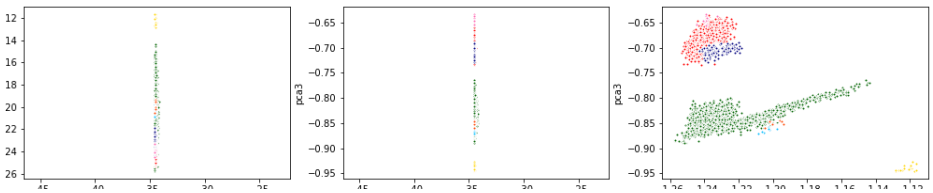


(a) K-means clusters using history_5 ARI=-0.016

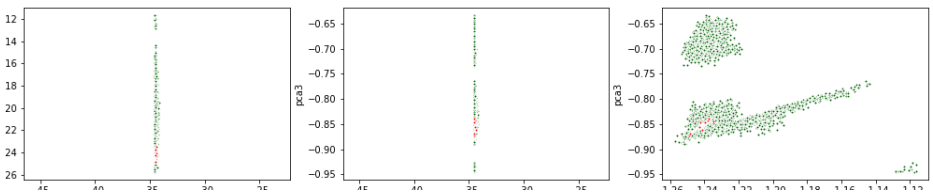


(b) Real label

Figure 4.8: K-means applied to history_5



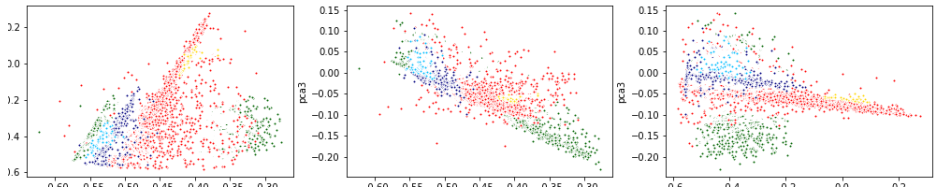
(a) DBSCAN clusters using kpi_1 ARI=-0.005



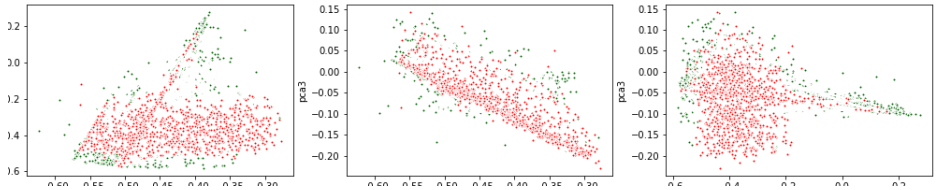
(b) Real label

Figure 4.9: DBSCAN applied to kpi_1

predict exactly two clusters, nevertheless the value of ARI is better than the one calculated using k-means. In order to improve performances of the algorithms used above, we try to find the best number of groups comparing the value of inertia for each different classification. Inertia is equal to the sum of squared distances of samples to their closest cluster center, and,

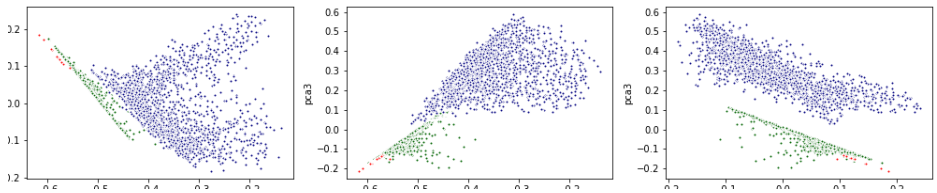


(a) DBSCAN clusters using kpi_4 ARI=0.029

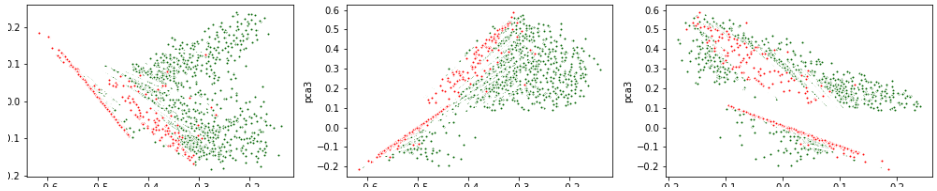


(b) Real label

Figure 4.10: DBSCAN applied to kpi_4



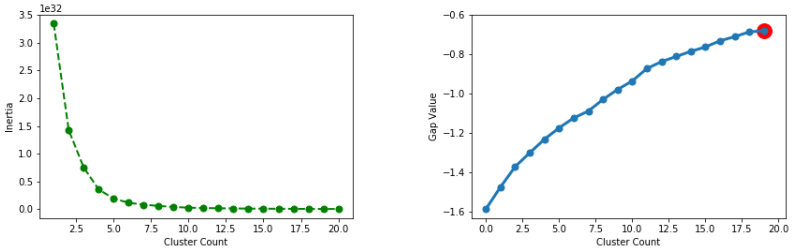
(a) DBSCAN clusters using history_5 ARI=0.168



(b) Real label

Figure 4.11: DBSCAN applied to history_5

as explained in the previous chapter, it usually decreases as the number of groups increases. So we have to plot the inertia for each value tested and see where the curve presents a 'kink', as represented in Figure 4.12(a). Since this procedure is not precise and could be misleading, we decide to use the GAPS method, but this method does not give good results, too. In fact,



(a) Example of inertia for number of cluster

(b) Example of inertia for number of cluster

the GAPS values go on increasing as the number of clusters increases(see Figure 4.12(b)). Considering that we do not obtain valid result from our cluster analysis we decide to do not investigate more on it and we pass to the part of manifold learning.

4.2.2 Manifold learning

As introduced in the previous chapter, the main issue of these methods is that they are computationally expensive. In fact, even if we select only a small part of our initial data set, we do not obtain any valid results for MDS and LLE. On the other hand, we manage to solve memory problem for isomap applying it to a subset of about 18000 observations. As it has be done for the cluster analysis, we project three subset of our features separately: accelerometer KPIs, positional KPIs and historical indicators. We reduce the dimensionality of the attributes set from 31 dimensions to 25 dimensions. In Figure 4.12 we report an example of the reconstruction errors, calculating to choose which hyperparameters we have to use to find the projections most similar to the original samples. Continuous, dotted and dashed lines represent the reconstruction errors when the number of neighbours is set respectively to 3,5, and 10. We can notice that the error decreases as the number of points in the neighborhood increases. However we have to consider that the computational cost increases, too. Finally in Chapter 4, during the supervised phase, we will train a classifier both on original data both on the projected one, since manifold learning algorithms should "clean" from noise our data set and so could give a more accurate

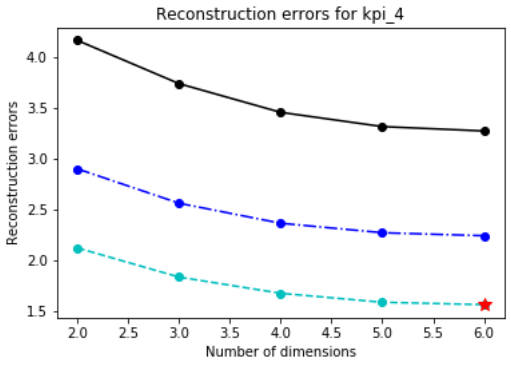


Figure 4.12: Reconstruction errors for kpi_4 varying the number of neighbours

model in the supervised part.

4.3 Summary of the results

In this chapter, we saw that our data set contains a lot of outliers and that fitting attributes distributions did not give valid results, that is why we used isolation forests. Another issue shown by this chapter is that clustering did not work on our data set, because false crashes and real alarms are not linearly separable. Moreover, we found out that the high number of samples composing our data set is a problem when we would like to apply algorithms expensive as regarding the computational cost, such as DBSCAN or multidimensional scaling.

Chapter 5

Supervised learning results

In this chapter we will present the results obtained training the classifiers presented before. After that, we will compare the precision of the statistical learning models fitted using original data with the ones found using original samples plus attributes derived by outliers detection, and with the ones obtained using manifold learning projections. Finally we will test our classifiers on a completely new data set, simulating the process that there will be in the Alfaevolution operation center.

5.1 Main results

For each model we calculate the percentage of false alarms and real crashes classified correctly and we collect our data in tables as the one shown in Table 5.1. In these tables we have six different columns, which represent:

- β: value of β used to train the model
- *Perc. false crash*: percentage of false alarms misclassified
- *Perc. crash*: percentage of real crashes misclassified
- *Algorithm*: classifier used to train the model
- *Data*: subset of kpis in the training set(see Table 3.1)
- *Avg*: average between columns 2 and 3

- *False Avg*: weighted average between columns 2 and 3, calculated as $0.90 \cdot Perc.falsecrash + 0.1 \cdot Perc.crash$

We calculate average because we want to find out which model classifies correctly the highest percentage of signals. On the other hand, since each call made by the insurance company in order to check if the driver needs assistance has a cost, we look for classifiers that have the highest precision for the false alarm class. We cannot order the models just using the percentage of false crashes classified correctly because there are some algorithms that, when are applied on the unbalanced data set, assign to each sample the label 'false crash'. This implies that these classifiers predict correctly all the false alarms, but they do not correctly classify any crashes. Hence, we decide not to consider them as valid models. Before analyzing the results obtained, we have to say that since gradient boosting and random forest algorithms are computationally expensive and have more hyperparameters to tune than the other models, we did not get valid results for these classifiers when applied on the unbalanced training set. As reported in Table 5.1, the first and the

β	*Perc. false crash*	*Perc crash*	*Data*	*Algorithm*	*Train*	*Avg*	*False Avg*
10	1.45%	2.31%	allh	abcw	1	1.88%	1.54%
10	1.45%	2.31%	allho	abcw	1	1.88%	1.54%
0.75	1.33%	2.48%	allho	gbc	2	1.91%	1.45%
1	1.29%	2.53%	allho	abcw	1	1.92%	1.42%
0.5	1.29%	2.53%	allho	abcw	1	1.92%	1.42%

Table 5.1: Best classifiers train on average

second best model use the adaboost classifier trained on the unbalanced data with $\beta = 10$, while the third best model is gradient boosting classifier, fitted on the other data set, using same KPIs, but with $\beta = 0.75$. On the other hand, the model which classifies the highest percentage of false alarms is the weighted decision tree, train on set 1, using allh data set and with $\beta = 2$(Table 5.2). Finally, the classifier with the lowest percentage of misclassified incident is the random forest trained on the balanced data set, with $\beta = 1$,

using the KPIs subset allhc(Table 5.3).

β	*Perc. false crash*	*Perc crash*	*Data*	*Algorithm*	*Train*	*Avg*	*False Avg*
2	0.63%	6.10%	kpih	dtcw	1	3.30%	1.17%
2	0.67%	5.33%	kpic	dtcw	1	3.25%	1.20%
2	0.63%	6.74%	allho	dtcw	1	3.53%	1.21%
2	0.67%	7.01%	allhc	dtcw	1	3.39%	1.22%
5	0.69%	2.89%	kpic	dtcw	1	3.40%	1.23%

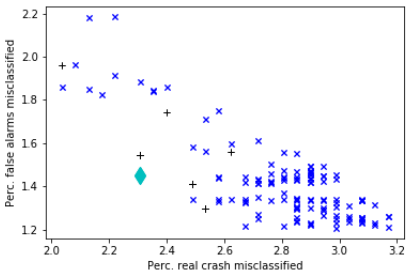
Table 5.2: Best classifiers comparing percentage false crash

β	*Perc. false crash*	*Perc crash*	*Data*	*Algorithm*	*Train*	*Avg*	*False Avg*
1e+15	16.14%	1.38%	kpi_1	abc	1	8.75%	14.66%
1e+15	16.14%	1.38%	kpi_1c	abc	1	8.75%	14.66%
1e+15	16.14%	1.38%	kpi	abc	1	8.75%	14.66%
1	15.98%	1.58%	kpi_1	abc	1	8.78%	14.54%
5	15.98%	1.58%	kpi_1	abc	1	8.78%	14.54%

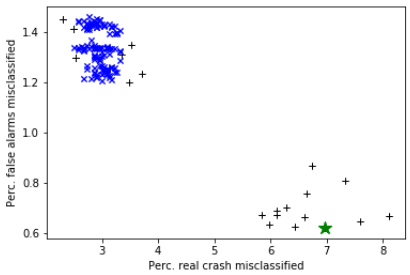
Table 5.3: Best classifiers comparing percentage crash

In Figure 5.1 we report graphically the results shown in the Tables 5.1, 5.2, 5.3:

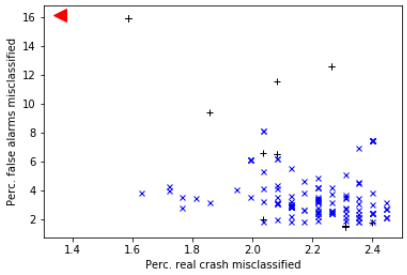
- the blue crosses indicate the performances of the algorithms train on balanced data set;
- the black plus represent the percentage of misclassified alarms using the unbalanced data set;
- the cyan diamond shows the best model on average;
- the green star evidences the algorithm that classifies correctly the highest percentage of false crashes;



(a) Best 150 classifiers on average



(b) Best 150 classifiers comparing media_false



(c) Best 150 classifiers comparing percentage crash

Figure 5.1

- the red triangle indicates the algorithm that classifies correctly the highest percentage of crashes;

Commenting these results, we could say that the classifiers trained on the unbalanced data set seem to have better predictive performance than the ones fitted on the balanced data set, when the samples are weighted. Nevertheless, we have to consider that the test set of the balanced data set contains twice of false alarms than the set used for testing performance of the unbalanced data set, and so this could be the cause of the lower accuracy of the models found using the second data set.

We decide to check which are the models trained on the balanced data set, in order to see if random forest and gradient boosting applied on the samples of adaboost give better results. As shown in Table 5.4 , we can notice that

β	*Perc. false crash*	*Perc crash*	*Data*	*Algorithm*	*Train*	*Avg*	*False Avg*
0.75	1.33%	2.48%	allh	gbc	2	1.91%	1.44%
10	1.33%	2.48%	allho	gbc	2	1.91%	1.44%
5	1.32%	2.57%	allh	gbc	2	1.95%	1.45%
1	1.33%	2.57%	allho	gbc	2	1.95%	1.45%
5	1.34%	2.57%	allho	gbc	2	1.96%	1.46%

Table 5.4: Best classifiers train on balanced data set

GBC[1] and RFC[2] have more predictive power than ABC[3], so we suppose that if we could run GBC and RFC on the first data set, we would have better results than the ones reported in Table 5.1. In the previous chapter, we explain that all the algorithms used, but neural networks, calculate how much a variable is 'important' in the fitted model. For this reason, we plot the features importances for the best classifiers described above(Tables 5.1, 5.4) in order to compare them. From Figure 5.2 and Figure 5.3, we

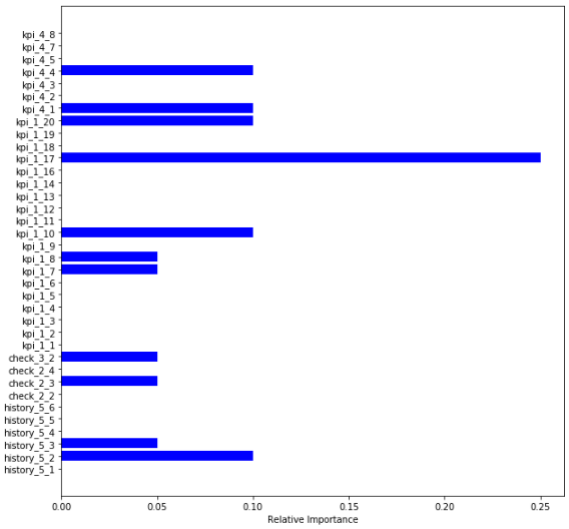


Figure 5.2: Feature importance of the best model trained on unbalanced set

[1]Gradient boosting classifier
[2]Random forest classifier
[3]Adaboost classifier

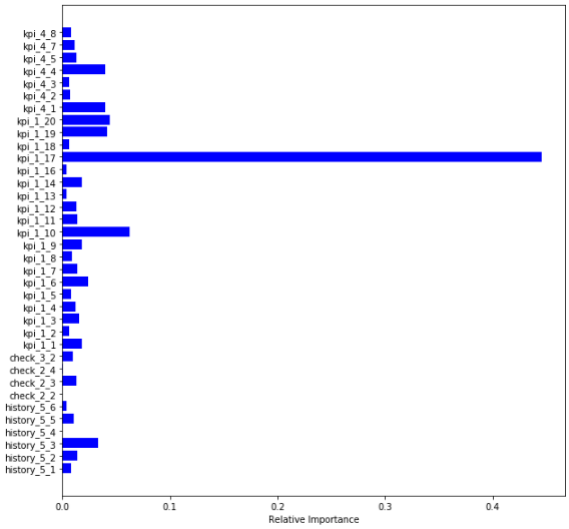


Figure 5.3: Feature importance of the best model trained on balanced set

can notice that ABC uses less features respect to GBC, that gives more or less importance to almost all the variables available. Nevertheless, the key performance indicators that influence the most both the classifiers are the same: kpi_1_17 is the most important one, followed by kpi_1_10 and kpi_1_20, kpi_4_1 and kpi_4_4. Moreover these plots show that the KPIs most used are the one referring to the accelerometer and that while ABC gives importance to historical data as the positional ones, while GBC consider less the KPIs history_5.

Next step is to control if the observations misclassified depend on the subset of KPIs used. Comparing the samples classified in the wrong way, using the same algorithms and changing only KPIs contained in the training set, we can notice that even if the models trained just considering the positional KPIs have worse predictive performance, they can predict correctly the target variable of some observations that are misclassifed by all the other data sets; this situation is evident especially when we take into account false alarms. Probably that is because when we consider all the KPIs, indicators referring to the accelerometer allow us to predict correctly the target variable of the most part of the samples, so they are given more importance, but doing this, models loose some information given by the positional kpis.

Finally, we notice that there are some accidents that are never predicted as real crashes. In order to analyze these samples, we plot the distribution of the most important KPIs both considering all the accidents, both taking into account only the one which are always misclassified. Figures 5.4 and 5.5 show that kpi_1_17 and kpi_1_12, which are two of the most important variables in all the trained model, assume smaller values for the most part of the samples whose target variable is never predicted in the right way. Probably this is problem is due to some faults of the black boxes.

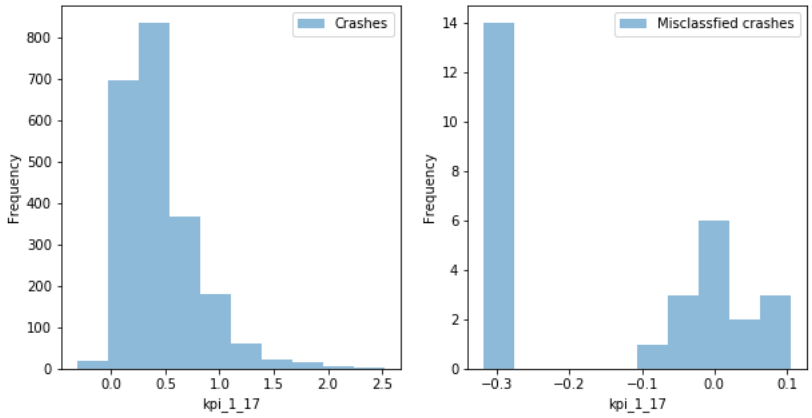


Figure 5.4: Distribution of kpi_1_17 for crashes and misclassified incidents

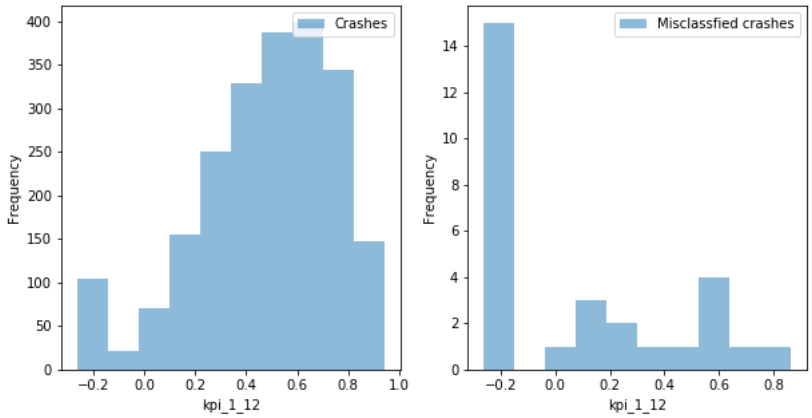


Figure 5.5: Distribution of kpi_1_12 for crashes and misclassified incidents

5.2 Outliers detection and supervised learning

As explained in Chapter 2, a pecularity of our data set is that it presents a lot of outliers, that could be identified using isolation forests. In this section we use the information about outliers to see whether we could improve predictive power of the models presented in the previous section. First of all, we apply isolation forests to our training sets for some kpis subsets(accelerometer kpis, positional indicators and historical data) and for each group we add to our data set a new dummy variable, that assumes value 1 when the sample is considered an outlier by the classification forest trained on that subset. After that, we fit the best model described in the previous chapter on the new data set containing dummy variables 'outliers_kpi_1','outliers_kpi_4' and 'outliers_history_5'. Even if new variable seems to have an impact on the

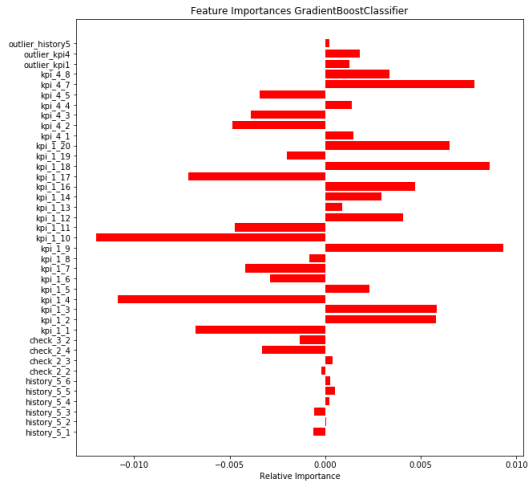


Figure 5.6: Feature importance differences between GBC trained on allh data and allh data plus outliers attributes

fitted models, as shown for example in Figure 5.6, the overall performances of the new classifiers are slightly worse than the ones of the original classifiers.

5.3 Manifold learning and supervised

Now we will compare the best classifiers described in Table 5.1 to the ones obtained applying our algorithms on the projections found using isomap in chapter 3. In Figure 5.7 we report the results obtained:

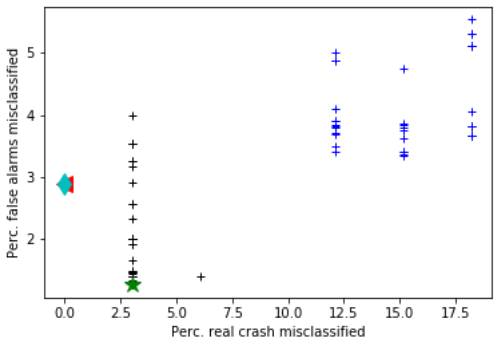


Figure 5.7: Predictive performance of algorithms using original data(black) and isomap projections(blue)

- the blue points indicate the performances of the algorithms that use isomap projections;
- the black dots represent the percentage of misclassified alarms; using the original data;
- the cyan diamond shows the best model on average;
- the green star evidences the algorithm that classifies correctly the highest percentage of false crashes;
- the red triangle indicates the algorithm that classifies correctly the highest percentage of crashes;

As shown in Figure 5.7, the original models have an higher predictive power, even if we supposed that isomap projection would give better results. Probably this is due to the fact that the algorithms that we choose to train our models are robust to outliers and collinearity. The only pros of manifold learning is that, reducing the dimensionality of our data set, we reduce the computational cost,too.

5.4 Comparison between models using new data

After finding the best models as explained in the previous sections, we test our algorithms on a completely new data set, that contains the black boxes alarms, registered during May and June. In Tables 5.5 and 5.6 we report the results of tests using all the pipeline implented on the new data set.

β	*Perc. false crash*	*Perc crash*	*Data*	*Algorithm*	*Train*	*Avg*	*False Avg*
10	1.94%	2.83%	allho	mpl	2	2.38%	2.03%
1e+15	1.30%	3.48%	kpih	gbcw	2	2.39%	1.52%
2	1.58%	3.34%	allho	abcw	2	2.46%	1.75%
1	1.58%	3.34%	allho	abcw	2	2.46%	1.75%
0.75	1.58%	3.34%	allho	abcw	2	2.46%	1.75%

Table 5.5: Best classifiers for new test set on average

β	*Perc. false crash*	*Perc crash*	*Data*	*Algorithm*	*Train*	*Avg*	*False Avg*
0.75	0.85%	4.49%	allho	gbc	2	2.67%	1.22%
0.50	0.85%	4.56%	allho	gbc	2	2.71%	1.22%
2	0.86%	4.61%	allho	gbc	2	2.74%	1.24%
1	0.86%	4.68%	allh	gbc	2	2.77%	1.24%
5	0.86%	4.60%	allho	gbc	2	2.73%	1.24%

Table 5.6: Best classifiers for new test set comparing media_false

We can notice that the models trained on the unbalanced data set perform worse than the model trained on the balanced set, when both are tested on the same test set. Moreover gradient boosting classifier is again the best model looking at media_false. Probably this is due to the fact that gradient boosting usually have a higher predictive power than all the other algorithms used, and that perform analysis on a balanced data set, even if it contains less samples, usually gives better results.

5.5 Summary of the results

In this chapter, we found out that in our problem we obtain statistical learning model more precise if we take into account our original samples, and not the projected ones calculated using isomap. Moreover we saw that the performances obtained are not affected by the presence of outliers among data. Finally we noticed that using gradient boosting classifier only on the balanced and smaller data set gives better results than using simpler algorithms on a bigger and unbalanced data set. We suppose that training gradient boosting classifier on the unbalanced data could fit a statistical learning model more precise than the one found in this work, but we did not have enough computational power to prove it.

Chapter 6

Conclusions

In this final chapter we will summarize what we did in this work and we will describe some of the possible future developments. After a brief introduction about machine learning and about the problem that we treated, we described the data set used to perform our analysis and how they are derived by Alfaevolution. Furthermore, we introduced the algorithms and the methodology used to: have a deeper knowledge of our data set, looking for linear correlation between features and investigate attributes distribution; perform data preprocessig and find noticeble differences between real and false crashes; fit a statistical learning model in order to predict if a signal refers to a false alarms or not. Finally we shown in details the results found applying all the algorithms introduced before.

The main issue to determine which model is actually the best is that we do not know the different prices of missing a false alarm or missing the prediction of a real crash. In fact, considering a model that classifies correctly all the accidents but that misclassified a lot of false crash, it could be not the optimal one, because calling too many clients is too expensive. On the other hand, a classifier too restrictive that does not predict correctly the most part of the accidents and that has very good precision as regarding false alarms is cheaper, but too many customers will be disappointed from the services, damaging the image of the insurance company. That is why, we were told by Alfaevolution, that they would like to keep the number of calls made during a month under the threshold of 25000; considering this constraint,

the best models on average are reported in Table 6.1. From this table we

β	*Perc. false crash*	*Perc crash*	*Data*	*Algorithm*	*Train*	*Avg*	*False Avg*
1e+15	1.30%	3.48%	kpih	gbcw	2	2.39%	1.52%
10	1.25%	3.70%	allh	gbcw	2	2.47%	1.49%
1e+15	1.21%	3.75%	allh	gbcw	2	2.48%	1.47%
1e+15	1.24%	3.81%	allho	gbcw	2	2.52%	1.50%
10	1.22%	3.84%	allho	gbcw	2	2.53%	1.48%

Table 6.1: Best classifiers for new test set constraining the number of calls

can understand that the model that gives the best precision, constraining the number of calls is Gradient boosting classifier, as noticed when we classify algorithms looking at media_false instead of media, even if in this case the samples are weighted. One of the possible further developments of this work could be rearranged all the steps developed here into an optmization model in order to find the classifier with highest precision and a number of call lower than a certain fixed threshold.

As explained in the previous chapter, analyzing the errors of each classifier, we notice that even if the models using only kpi_4 have the lowest overall precision, they are able to recognize some false crashes misclassified from all the other models. Therefore, we suppose that in order to decrease the number of calls that have to be done we can use a simply but effective strategy. First of all we fit the best model using the subset of features that ensures the highest predictions rate and another model taking into account only positional KPIs. After that we predict samples labels using the former model, then we consider only the sample classified as accident and, finally, we predict another time labels using the latter algorithm. Using this procedure, we recognize less real crashes, but we classify correctly thousands of false alarms, too. Finally, we believe that creating and using a new balanced training set, where the number of samples referring to real accidents is higher, we will generate model with a higher predictive power.

Bibliography

[1] Freund, Y. & Schapire, R. E. (1996) *Experiments with a New Boosting Algorithm. In Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, ICML'96, pp. 148–156. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN 1-55860-419-7.

[2] Friedman, J., Hastie, T. & Tibshirani, R. (2000)*Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors).* Annals of Statistics 28, 337–407. doi:10.1214/aos/1016218223.

[3] Tibshirani, R., Walther, G. & Hastie, T. (2001) *Estimating the number of clusters in a data set via the gap statistic. Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63, 411–423. doi:10.1111/1467-9868.00293.

[4] Chen, T. & Guestrin, C. (2016) *Xgboost: A scalable tree boosting system. In Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 785–794. New York, NY, USA: ACM. ISBN 978-1-4503-4232-2. doi:10.1145/2939672.2939785

[5] Hastie, T. J., Tibshirani, R. J. & Friedman, J. H. (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* New York, USA: Springer, second edition. ISBN 9780387848587

[6] Bishop, Christopher M.,(2006) *Pattern Recognition and Machine Learning (Information Science and Statistics).* Berlin, Heidelberg: Springer-Verlag. ISBN 0387310738

[7] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.