POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea Magistrale in collaborazione con Tierra S.p.A.

Predictive Maintenance for off-road vehicles based on Hidden Markov Models and Autoencoders for trend Anomaly Detection

Relatori
Prof. Francesco Vaccarino
Prof. Luca Cagliero
Tutor aziendali
Dott.ssa Lucia Salvatori
Dott. Elvio Amparore
Dott. Riccardo Loti

Candidato
Lorenzo Perini
s253570

Anno Accademico 2018-2019

« In the long run, all machines break down. »

[John Maynard Keynes]

« Predictive modeling generates the entire model from scratch. All the model's maths or weights or rules are created automatically by the computer. The machine learning process is designed to accomplish this task, to mechanically develop new capabilities from data. This automation is the means by which Predictive Analytics builds its predictive power. »

[Eric Siegel, *Predictive Analytics: The Power to Predict Who Will Click, Buy, Lie, Or Die*]

Contents

List of Figures

List of Tables

Acknowledgement

Per prima cosa un grande ringraziamento al Prof. Vaccarino, relatore di questa tesi di laurea, sia per il grande contributo che mi ha fornito in questi mesi dandomi la conoscenza teorica sufficiente ad iniziare e concludere questo elaborato, che per la disponibilità, la critica costruttiva e il sostegno che hanno fatto sì che portassi a termine il lavoro. Nonostante tutti gli impegni, mi ha sempre concesso un'ora di tempo per revisionare la tesi, confrontarci e arricchire le mie conoscenze. Senza la sua pazienza e la sua guida non sarei riuscito a concludere un percorso di studi in tempo per la sessione di laurea di Luglio 2019. Insieme a lui, voglio ringraziare anche il Prof. Cagliero per l'aiuto decisivo nella fase conclusiva.

Un meritato e sentito ringraziamento va all'azienda Tierra S.p.A. e a tutto il team di "applied research & data analytics". Per merito di Riccardo ho iniziato l'avventura, durata poi 5 mesi, in quest'ottima azienda che mi ha dato la possibilità di terminare il mio percorso di studi applicativo con un lavoro di ricerca applicata a dati reali. I miei ringraziamenti vanno soprattutto a Lucia e Elvio, i quali mi hanno seguito fin dall'inizio dandomi spunti di riflessione e aiutandomi a trarre ispirazione dai risultati raggiunti per andare avanti. Sicuramente è merito loro se, nonostante le premesse non molto convincenti, alla fine il lavoro è terminato con un buon risultato positivo. Da Lucia ho colto la grande importanza che ha la statistica all'interno del lavoro di un Data Scientist, grazie ai mille confronti soprattutto durante la pausa pranzo. Da evidenziare che, grazie a lei, ho conosciuto ed iniziato a vedere ovunque le Heat map, come si evince dalla quantità di grafici presenti nella tesi. Da Elvio ho invece percepito la bellezza del Machine Learning, tanto

da rafforzare l'idea che il mio futuro professionale comprenderà questo ambito. É merito suo se ho provato e sto continuando a fare richiesta per un Ph.D. all'estero in una delle principali università in Europa, anche se sicuramente a detta sua "lui non conta niente". Possono passare mesi, anni o anche decenni, ma il bigliettino da visita di Elvio resterà sempre sopra la lavagna dell'ufficio 17.
Inoltre, vorrei ringraziare il mio ex collega tesista di ufficio Calogero per avermi lasciato, dopo una lunga lotta durata qualche settimana, l'intera lavagna per scrivere i miei risultati. Dopo essere passato da tesista a dipendente, ha avuto la grandissima responsabilità di riavviare Jupiter ad ogni mia richiesta. Questo lo ha obbligato a lavorare anche alle 23 : 30, quando era disteso comodo sul divano di casa ad oziare. Spero che lui riesca nell'intento di "far vivere così" il nuovo tesista Antonio, a cui suggerisco di cambiare mood (almeno passa a "non si può *quasi* vivere così").
Un grazie di cuore anche ad Andrea M., che è stato essenziale per la scelta dei dati da utilizzare. Il suo intervento è avvenuto nel momento in cui l'oscurità dei dati di Tesmec stava prendendo il sopravvento sul mio modello. Per merito suo e di Storti S.p.A., questo lavoro ha avuto un esito semi-positivo (evitiamo di sbilanciarci troppo!). E pensare che Andrea aveva anche previsto per me 6 mesi da passare in solitudine insieme alle mucche a Verona!

Vorrei ringraziare i miei colleghi del corso di laurea di Ingegneria Matematica, sia per il tempo trascorso a studiare insieme che per le rilassanti serate che mi hanno permesso di distaccarmi dalla pressione della laurea. In particolare un grazie ad Andrea per l'insistenza e la caparbietà con cui ha continuato a chiedermi di finire e mandargli il lavoro sin dal primo giorno di tesi, a Silvia per la positività che porta con sè e per la capacità di rendere chiare le frasi che Andrea usa quando mi avvisa delle serate, e a Lucrezia per le mille e passa storie sulla sua amata Forlì che, a quanto pare, è colma di storie inimmaginabili.

Un grandissimo ringraziamento anche a Serena, Giorgia, Elena, Margherita, Max, Daniele, Jack, Seba e gli altri per tutte le serate che definirei stimolanti, in

quanto passate all'insegna della birra e del biliardino. Grazie alla risata isterica di Serena, alla follia che condivide con Giorgia, al romagnolo profondo di Marghe, alla conoscenza di qualsiasi serie TV e film di Max, al cinismo estremo e pungente di Daniele, all'apertura e alla chiacchiera facile di Jack e alla convivialità di Seba, la cui casa è sempre aperta, le feste trascorse insieme a voi sono state le migliori. Inoltre un grandissimo grazie a Serena per avermi fatto capire cosa provano i telespettatori delle telenovelas, raccontandomi avvenimenti immaginari, simbolici e reali di mille e passa scene amorose per almeno un anno intero.

Passando ora alla famiglia, ringrazio prima di tutto i miei genitori per aver reso possibile questa bellissima esperienza passata a Torino. So che sono stati due anni brevi e pieni di sacrifici (soprattutto finanziari), ma penso e spero che ne valga la pena. Siete e sarete sempre i miei primi due motivatori. Da voi ho sempre trovato la spinta necessaria per buttarmi verso una carriera il più brillante possibile. Alla zia e ai nonni (Antella e Sorgane) mando un caloroso abbraccio: la domenica a pranzo è sempre stato e sempre sarà un momento di ritrovo fantastico in famiglia. Un ricordo speciale va al nonno Franco e al nonno Ade. Nonostante siano passati poco più di 7 anni dall'ultima volta in cui ci siamo parlati, ricordo come fosse ieri le giratine in bici e nei boschi con il nonno Franco. A te, nonno Adelindo, un grande saluto. É grazie a te se da oggi punterò sempre più in alto, grazie agli ultimi suggerimenti che oggi sono diventati il mio motto: sempre meglio. Un imbarazzatissimo saluto e ringraziamento a mio fratello Niccolò e alla Benedetta, da cui ho trovato sostegno e apprezzamento. Se non fosse per la Benedetta non avrei nemmeno scritto i ringraziamenti. Meno male mi hai tartassato per mesi!
Infine un profondo ringraziamento ai miei parenti di Sesto Fiorentino e di Grosseto. Nonostante ci vediamo veramente poco, è sempre bellissimo festeggiare in famiglia!

Ai gemelli diversi Sandro e Fabio, al perenne "AFK" Dario, al saggio e antiquato Monto, al brillante Maestro voglio mandare un profondo ringraziamento, anche se in realtà non hanno contribuito minimamente alla stesura della tesi. Scherzi a parte, i fine settimana trascorsi con voi sono i migliori, soprattutto quando il

Maestro prende l'iniziativa sparando idee assurde, tipo entrare in casa del Monto (alla fine ci siamo riusciti!). Non so veramente come avrei fatto senza la perspicacia di Sandro, l'abilità da neo agente immobiliare di Fabio, l'energia di Dario, l'intraprendenza del Monto e l'umiltà del Maestro. Mi raccomando ragazzi, la prossima volta non aspettate la laurea per salire nella città dove ho vissuto 2 anni e fatemi visita almeno entro la fine del primo anno (non ci credo nemmeno io in questa frase, tranquilli). Un saluto anche ai due super, uno un po' sparito mentre l'altro appena ritrovato, anche se con la sua fantastica giacca sembra un esattore delle tasse e/o un testimone di Geova. Un particolare ringraziamento va alla miglior designer nel mercato Emma, che mi ha fatto un modellino 3d pazzesco dell'autoencoder che userò come bomboniera. Oltre a lei ci tengo a ringraziare la poetessa e neo cantante Ilaria e la tremendina, ecologista e imprenditrice Giada per le serate passate dal Maestro e per l'aria di gioventù che hanno portato al gruppo. Anche tutti gli altri (Sere, Lavi, Costi, Petru, Ele, Angelo, ecc. . .) nonostante non siano mai venuti da me a Torino, meritano un ringraziamento per avermi influenzato chi più e chi meno e aver contribuito a raggiungere questo stupendo traguardo.

Infine, ma non ultima ovviamente, un amoroso grazie va alla Lavi per avermi sopportato in tutti questi anni e non aver fatto troppe storie sul trasferimento a più di 400 km da casa. Ormai la distanza non fa per niente paura, visto che ci sentiamo quasi ogni giorno, anche se non appena ti metti al telefono a guardare i social l'attenzione diventa praticamente nulla! Sono così fiero del percorso di crescita che stiamo avendo insieme e vedrai che prima o poi tornerò a vivere a Firenze (forse). Volevo inoltre ringraziarti per le conoscenze di matematica che mi hai passato in questi ultimi mesi, nel senso che mentre ti aiutavo a studiare imparavo cose nuove! Scherzi a parte, sei il miglior sostegno che chiunque possa desiderare e sono felice che tu approvi tutte le mie idee, togliendomi un grosso peso ogni volta.

Dopo tutti questi ringraziamenti, qualsiasi sia il mio futuro a breve e lungo termine, spero di rimanere sempre in contatto con tutti voi, perché siete stati fondamentali per farmi arrivare dove sono adesso.

Chapter 1

Introduction and relative work

Off-road vehicle maintenance is getting increasingly important as the unplanned stops might significantly damage the entire work delaying the process beyond the limits. Traditional systems are gradually improved with new planned maintenance, according to some statistical and deterministic results. From this point of view, it is relevant to make a clear division between preventive maintenance and predictive maintenance. The former is commonly used and it consists of replacing periodically vehicle components. It is a policy where, independently of the real status of the unit, maintenance actions are applied according to the vehicle age. For this reason, a few vehicles are repaired in time while others fail before their maintenance. With the entry into play of artificial intelligence and IoT, predictive maintenance methods are getting more and more central topics. Through Machine Learning and Deep Learning approaches is currently possible to have an accurate prediction of failure or errors inside the system with up to few days in advance. In fact, predictive maintenance determines the general condition of the machine by monitoring its values and then it predicts with high accuracy if a failure is going to occur and, some times, even when it will happen. In addition, through the behaviour of each component, the model is able to claim even which part shall be repaired.

All data are collected from the devices of the company Tierra S.p.A., which operates in the IoT sector. The internet of things is a system of interrelated computing devices, mechanical and digital machines. It is also related to the ability to

transfer data over a network without requiring human interaction. So, the structure of data is made complicated by the interaction between machines which fill the data with noise. As a result, more accurate data cleaning techniques have to be applied in order to reach a reasonable accuracy score.

This thesis investigates unsupervised and supervised methods for predicting vehicle maintenance, meaning that we predict if a diagnostic message is going to occur. In order to achieve this goal, we are going to use probabilistic methods based on the concept of Markov chains, distance and tree based algorithms, and finally deep neural networks. From each of these methods pros and cons will be discussed, together with its statistical scores evaluating the real performance into the problem.

Final results confirms that predictive maintenance is not something we don't need to, but it could make people save time and money for real. By using the artificial intelligence, we will build a model reaching about 81% of F_1 score in the prediction of diagnostic messages occurrences. The model figures the trend out and perceives the closeness to the failure by claiming faulty values with at least 30 hours in advance.

The thesis is divided into three parts and it goes as follows. The first one is data preparation and it involves all the passages from the analysis of many rude datasets to the building of the right one as a base of our model [HPL02; A+16; Pry14; BDC02]. In this chapter we will find how to merge two or more datasets in order to achieve the main goal. So, the two most important database are deeply analyzed in order to build a final variable response which is a binary and deterministic label based on the timestamp of each row.

The second part involves the so called feature selection, where all the present parameters are reviewed and, through a comparison between each variable and the response label, just the set of the most significant for the rest of the analysis are selected by 4 methods. The first method is related to the *Mutual Information* and the *Kullback–Leibler* pseudo-distance [Mur17; CT91] which compares the distributions of data. It highlights an evident weak dependence among features and response variables. The second one is about *Recursive Feature Selection* with *Support Vec-*

tor Classifier [CL11; YZ15; Guy+02], which makes a ranking of the variables from the first relevant to the last one. Then, we apply a tree-based method based on Importance of features called *Extremely Randomized Trees* [GEW06]. The last one is a distance-based method and it considers *Dynamic Time Warping* distance to create many time series and to get out a score from them [Fur08; KP01; Fol+18; SC07; SZZ15; Rak+12]. With our kind of data, just one feature (with its moving average) will remain after the hard (in the sense of high thresholds) feature selection we impose. This feature is the one which is directly connected to the failure we want to predict, because it monitors directly the values that cause the damage.

The last part provides models to predict a maintenance need. The first one is a probabilistic model called *Hidden Markov Model* [DD99; Sal05; KD; Sta04; Rab89; Ye+16; YXC94], that creates k hidden states and each one of those contains a different probability to detect an error. It can be used as a trend anomaly detector to find out when values start to be anomalous. It reaches scores around $60 - 65\%$. The second one is called *kNN Outlier Detection* [RRS00; Ota+13] and it is a distance-based model which is able to detect outliers. It has scores around 70%, showing it limits under noisy data. The next model is called *Isolation Forest* [LTZ08; LTZ12] and it is based on sequential cuts of the data intervals in order to isolate each point. Looking at the amount of cuts needed for each data, it emits an anomaly score to classify the test set. The results are about 78%. Then we will apply *Autoencoders Neural Networks* [GE18; PAD18; Che+17] which is the best model from the point of view of performances, reaching $79 - 80\%$ of precision and recall with stratified cross-validation [ZM00]. The model is a neural network and it destroys data by reducing their dimensions and then it reconstructs all the points to compute the reconstruction error. Through this score each point is classified as normal point or pre-error point (faulty). The final method is a two step model called *Autoencoder on Hidden Markov Model*, which increases the interpretability of Autoencoders and it makes the model more adaptable. The results are slightly better than the Autoencoder, achieving $80 - 81\%$ of precision and recall.

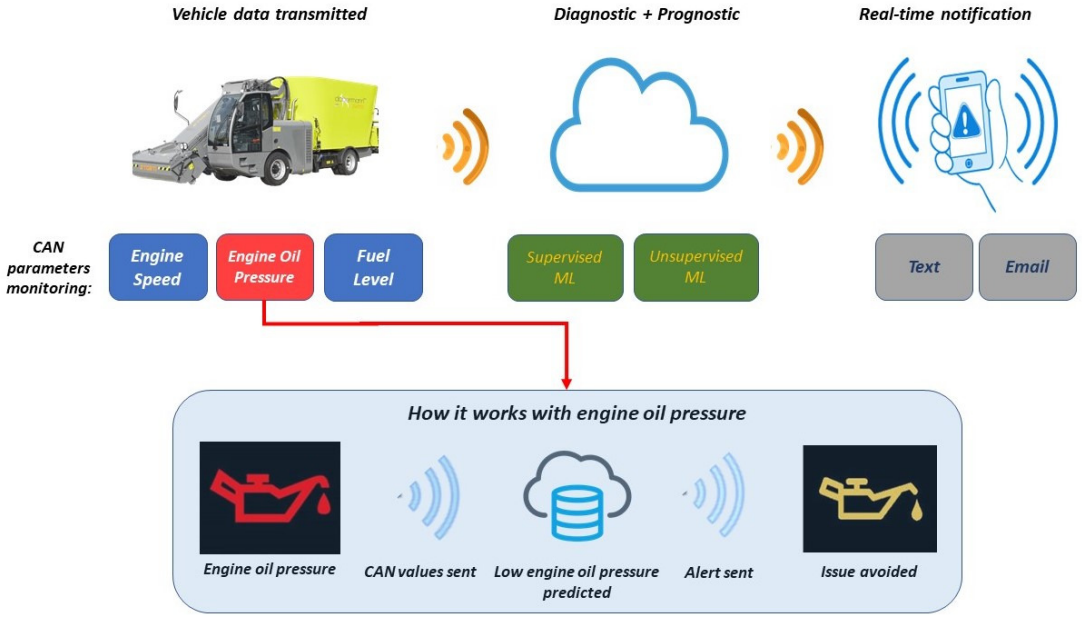


Figure 1.1: Predictive maintenance for off-road vehicles.

Images in Figure 1.1 are from
https://www.drivingtesttips.biz, https://www.storti.com,
http://itblueprint.ca, https://www.iconfinder.com,
https://www.internetpost.it, https://www.restroomalert.com.

Chapter 2

Predictive-Oriented dataset construction

2.1 Introduction to data preparation

The first part of data anlytics and, somehow, the most delicate and important part, is called data preparation. In fact, in order to achieve our task, having a precise and perfect dataframe is very relevant. Since our goal is to create a model for the prediction of the DM1s (see section 2.3.1), at the beginning we have to look for which DM1 (or, eventually, which DM1s) is relevant and if it's worth to make a deep analysis of it. Basically, this part should join together statistical skills and domain competences. For what concerns the domain, it has been useful to have organized a few meetings with the technician responsible for the sector. On the other hand, the statistical part concerns a shallow analysis the CAN parameters (see section 2.2.1) in order to select only those messages which are relevant for the work. As a result, one dataset reporting all the cleaned data informations about DM1s and one related to CAN messages are created. Then, the remaining part has the aim to build the final dataset by merging, somehow, both the two previous datasets.

2.2 Controller Area Network (CAN) parameters analysis

2.2.1 What is a CAN message

The Controller Area Network (CAN) is a serial communication protocol able to manage with high efficiency real time distributed control systems, with a high level of security and integrity of the transmitted data (SAE J1939 protocol, see [HPL02]). It is a vehicle bus standard and it is planned to allow devices to communicate with each other without using a host computer. The CAN messages are generated at a high frequency (up to 100 Hz) and are gathered by a controller, where they are collected and processed. In a CAN network, many short messages like temperature, pressure or RPM are broadcast to the entire network, which provides for data consistency in every node of the system.

All the vehicles we consider for the analysis may have from 20 to 40 electronic control units for various subsystems. Basically, various sensor inputs from around the vehicle are collated via the CAN bus to determine and track the current parameters of the unit. Through an analysis of those inputs it is possible to make some predictions, based on the kind of messages communicated. At each kind of parameters it corresponds a different code called SPN (Suspect Parameter Number), which identifies completely the CAN parameter. Basically, one SPN is a code which stands for one controlled parameters. Two different SPNs monitor two different parameters of the unit.

Fixed a unit, the granularity of the messages is about 10 minutes. It means that every 10 minutes (more or less) one value for each SPN is broadcast to the central unit control and it corresponds to a new line in the initial database (one per SPN). It is important to make clear the main negative aspect of the way these CAN messages are recorder: since every 10 minutes one signal reaches the server, inside these temporal gap many messages are saved (about 60.000 per 10 minutes). Then, in order to create just one line inside the dataframe, one of 4 statistics are computed: max, min, average and last. This procedure is not under our control

and we feel that it will be a hard obstacle to overcome. Currently, since SPN 100 will play a relevant role in the whole work, we want to make clear that its statistic is the "last". This means that unit broadcast only the last value of the sample of SPN 100 among all the observations inside the 10 minutes.

For this first work, two task should be achieved: the first one is to figure out if there is a sort of regularity inside those parameters, whereas the second one concerns a more specific and deep study in order to point out an eventual general behaviour in common among all the units.

2.2.2 CAN data exploration

At the beginning let's consider all the units present inside the database of the same tenant, which is in this case Storti S.p.A.[1] The Tierra S.p.A. dataset contains information on 8 different unit models, belonging to the same Dobermann series. The total number of vehicles is 22 and each unit has broadcast a different amount of CAN messages, both in the sense of variety and quantity.

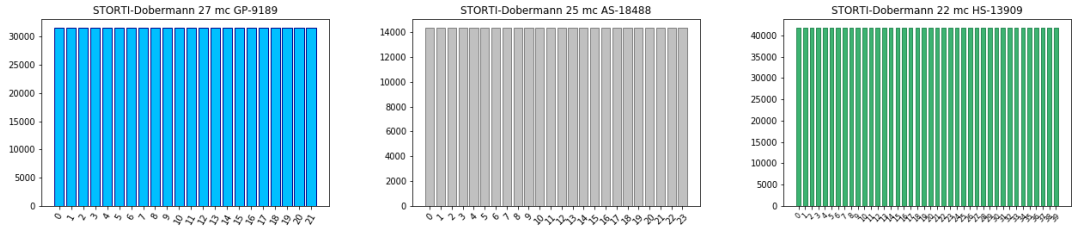


Figure 2.1: Amount of CAN messages for three units with 22, 24 and 40 SPNs respectively.

Looking at the Figure 2.1 we should notice that, trusting us that it is a significant sample of the entire group of vehicles, all the units broadcast more or less the same amount of observations for each SPN. Fixed a unit, the max difference between the largest and the smallest amount of data (SPN by SPN) is two orders smaller than the count value. So, we could claim for sure that just a small amount of data is lost (less than 1% of data for each unit). This regularity makes the whole

[1]Storti S.p.A. is a leading company in the production of animal feed machines with Unifeed technique.

analysis be more robust and the conclusion, if there exists a positive result, more accurate. On the other hand, as reported in Figure 2.2 the total number of CAN messages among the units is very different: it goes from about 90 timestamps for a less used unit, to 80.000. During this work we have to keep in mind this peculiarity, because the length of the time series may change a lot. Mostly, for the construction of the predictive model we have to remember that since part of the approach that we will use is related to sequenciality of temporal data, the length of a sequence may have a different impact on the results.

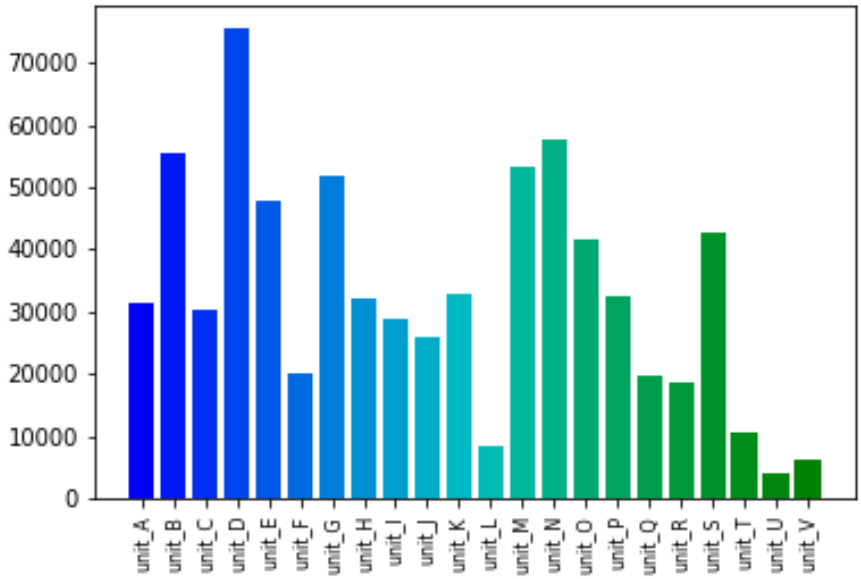


Figure 2.2: Amount of CAN messages per unit.

2.2.3 Clustering units according to their general behaviour

In this second preprocessing part, the goal is to find out if there exists a substantial division inside the Storti units in order to treat each group in a distinct way for the rest of the analysis. In fact in case we find out the presence of many groups of vehicles, it could be reasonable to consider for the model each group separately, in order to achieve a higher level of accuracy by not wasting intergroup informations.

The most used methods to create distinct groups of data involve the clustering techniques. Clustering is the process of grouping similar entities together. The goal of this unsupervised machine learning technique is to find similarities within the data and to group similar data points together, according to a fixed measure.

We begin by defining some notation. Let $C_1, \ldots, C_K$ denote cluster sets containing the observations. These sets satisfy two properties:

- $C_1 \cup C_2 \cup \cdots \cup C_K = \{x_1, \ldots, x_n\}$. In other words, each observation belongs to at least one of the K clusters.
- $C_k \cap C_{k'} = \emptyset$ for all $k \neq k'$. In other words, the clusters are non-overlapping: no observation belongs to more than one cluster.

To implement a correct version of clustering, we should before decide which distance might be the best one, in the sense of interpretability and operation. It is also necessary to hold the structure of sequential events: each observation has a non zero correlation with the next one, so that it could be self-defeating to treat data as they were independent. In addition, it might be wrong to make that assumption from a mathematical point of view. After a long analysis of which distance could be the right one, we finally conclude that it's the Dynamic Time Warping the most suitable and complete distance. For the explanations of how it works, we refer you to the chapter 4.6.

For the sake of simplicity, we take into account the two most popular clustering algorithm: K-Means and K-Medoids. The input to these algorithms are randomly distributed data points and, through them, clusters are generated according to their similarity. The theoretic results demonstrate that K-Medoids, as compared to K-Means, is better not only in terms of execution time, but even because it is sensitive to outliers and it reduces noise, since it minimizes the sum of dissimilarities of data objects, as it is shown in [A+16].

K-Means

The K-Means algorithm is a well-known partitioning method for clustering. K-Means clustering method groups data according on their closeness to each other by

using the *Euclidean distance*. As we can see in the algorithm 1 where the pseudo-code is presented, it starts getting k as an input parameter and partitioning a set of n object inside k clusters. The mean value of data is taken as similarity parameter to form cluster centroids. Then, the instances of the database are relocated to the cluster represented by the nearest centroid in an attempt to reduce the square-error (2.2). This relocation of the instances is done following the instance order. If an instance in the relocation step changes its cluster membership, let's say between the $s-$th and $t-$th clusters, then the centroids of the clusters C_s and C_t and the square-error should be recomputed. This process is repeated until convergence, that is, until the square-error cannot be further reduced. In other words, it means that the algorithm will stop once no instances changes its cluster membership (Figure 2.3).

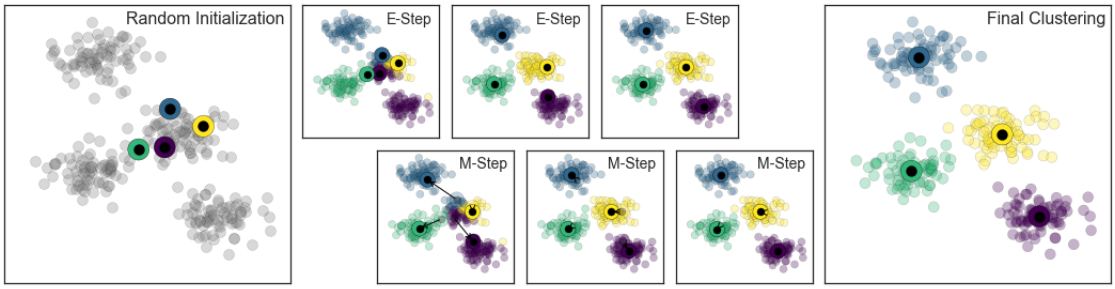


Figure 2.3: K-Means algorithm example. The image is taken from the site https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html.

input : Number of clusters K and data-points $\{x_1, \ldots, x_n\}$.
output: Clusters $C_1, \ldots, C_K$.

Select K points as the initial centroids;
while *the centroids don't change* **do**
    Form K clusters by assigning all points to the closest centroid;
    Recompute the centroid of each cluster;
end
return Clusters $C_1, \ldots, C_K$.

Algorithm 1: K-Means algorithm pseudo-code.

Almost all partitional clustering methods are based upon the idea of optimizing a function F referred to as clustering criterion which, hopefully, translates

one's intuitive notions on cluster into a reasonable mathematical formula. The function value usually depends on the current partition of the dataset in groups $\{C_1, \ldots, C_K\}$. That is:

$$F : \mathcal{P}_K(\Omega) \to \mathbb{R}, \tag{2.1}$$

where $\mathcal{P}_K(\Omega)$ is the set of all the partitions of the dataset $X = \{x_1, \ldots, x_n\}$ in K non-empty clusters. Each x_i of the n elements of the database is a N-dimensional vector. Concretely, the K-Means algorithm finds locally optimal solutions using the sum of the l^2 distance between each element and its nearest cluster center (centroid) as clustering criterion (F). This criterion is usually referred to as square-error criterion. Therefore, it follows that:

$$F\left(\{C_1, \ldots, C_K\}\right) = \sum_{i=1}^{K} \sum_{j=1}^{K_i} \left\| x_{ij} - \bar{x}_i \right\|_2, \tag{2.2}$$

where K is the number of clusters, K_i the number of data in the cluster i, x_{ij} is the $j-$th observation of the $i-$th cluster and $\bar{x}_i$ is the centroid of the $i-$th cluster, which is defined as

$$\bar{x}_i = \frac{1}{K_i} \sum_{j=1}^{K_i} x_{ij} \quad \forall\, i = 1, \ldots K. \tag{2.3}$$

Despite being used in a wide list of applications, the K-Means algorithm has a few cons. Some of the most important drawbacks are listed below:

- As many clustering methods, the K-Means algorithm assumes that the number of clusters K in the dataset is known. Obviously, it is not necessarily true in real-world applications;
- As an iterative technique, the K-Means algorithm is especially sensitive to initial starting conditions (initial clusters);
- The K-Means algorithm converges finitely to a *local minimum*. The running of the algorithm defines a deterministic mapping from the initial solution to the final one. Although there is no guarantee of achieving a global minimum, at least the convergence of the algorithm is ensured.

K-Medoids

The K-Medoids algorithm is used to find medoids in a cluster which is its centre located point. K-Medoids is more robust than K-Means because in K-Medoids the aim is to minimize the sum of dissimilarities of data, according to a fixed dissimilarity measure, whereas K-Means uses sum of squared Euclidean distances to look for similarities of data. This distance metric reduces noise and outliers.

In general K-Medoids algorithm is used to reduce the drawbacks of K-Means, since it is based on medoids instead of centroids. For each cluster, the medoids is its the data point which is most centrally located. The main difference between centroids and medoids is that the first one are imaginary data points, which has as components the mean of those of the data points within the same group. On the other hand, the medoids are real data points. Because of this distinction, K-Medoids doesn't suffer so much from the presence of outliers. In fact it can happen that adding a data point to a cluster, no matter how much it is different from the others, the previous centroid doesn't change. The pseudo-code algorithm is reported in algorithm 2. At the beginning, medoids are selected randomly from the n data to form the K cluster centers. Then the other remaining data are assigned to the closest medoid, creating clusters based on the measure of dissimilarity. After that, we have to process all the data of each cluster to find new medoids computing the total distance between each point and all the others. The new medoid is the one which minimizes that total distance. After finding the new centers let's reallocate all the data to the clusters. The location of medoids changes during each iteration. When they stop changing, the result is achieved and the clusters $C_1, \ldots, C_K$ are formed.

Silhouette Analysis

Before applying clustering techniques, we need to determine the best value of K, which is the number of clusters. For this approach the most used method is the silhouette analysis.

Silhouette analysis can be used to study the separation distance between the resulting clusters, and the Silhouette score takes values in a range of $[-1, 1]$. Silhou-

input : Number of clusters K and data-points $\{x_1, \ldots, x_n\}$.
output: Clusters $C_1, \ldots, C_K$.

Calculate the distance between every pair of all objects based on the chosen dissimilarity measure (d);
Select K points as the initial medoids;
while *the centroids don't change* **do**
  Find a new medoid of each cluster, which is the object minimizing the total distance to other objects in its cluster;
  Update the current medoid in each cluster by replacing with the new medoid;
  Assign each object to the nearest medoid and obtain the cluster result;
end
return Clusters $C_1, \ldots, C_K$.

Algorithm 2: K-Medoids algorithm pseudo-code.

ette coefficients near +1 indicate that the sample is far away from the neighbouring clusters and so that the partition is very accurate, while values close to 0 indicates that the sample is on, or very close to, the decision boundary between two neighbouring clusters. Finally, negative values indicate that those samples might have been assigned to the wrong cluster.

After that data are grouped via K-Means or K-Medoids into K clusters, for each point x_i, let $a(i)$ be the average distance between x_i and all other data within the same cluster. We can interpret $a(i)$ as a measure of how well x_i is assigned to its cluster (the smaller the value, the better the assignment). We then define the average dissimilarity of point x_i to a cluster C as the average of the distance from x_i to all points in C.

Let $b(i)$ be the smallest average distance of x_i to all points in any other cluster, of which x_i is not a member. The cluster with this smallest average dissimilarity is said to be the "neighbouring cluster" of x_i because it is the next best fit cluster for that point. So it is possible to define a silhouette as:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}. \tag{2.4}$$

The average $s(i)$ over all points of a cluster is a measure of how tightly grouped

all the points in the cluster are. Thus the average $s(i)$ over all data of the entire dataset is a measure of how appropriately the data have been clustered.

2.2.4 Unsupervised results

Because of the reasons explained above, the results we will take care about concern the K-Medoids clustering with Dynamic Time Warping as measure of similarity/dissimilarity. Obviously, the dataset consists in n data points $\{x_1, \ldots, x_n\}$ where each one represents all the CAN messages received in a precise timestamp, one for each SPN. But what we want to group are not the basic data points. Since each point has a label, which indicates what's the related vehicle, it's possible to consider macro data points where each observation corresponds to one unit. In doing it, every points are composted by many CAN messages, so that they are properly multidimensional time series (3).
Basically, the task is to cluster units having similar CAN value behaviours. In order to build those groups, we can compare time series through the DTW distance. The first step is to arrange distances into a matrix, in order to speed up the K-Medoids algorithm, as reported in the first part of 2.1.

```
1  distanceMatrix = [[0 for x in range(len(df))]
2                           for y in range(len(df))]
3
4  for i in range(0, len(df)):
5      for j in range(i+1, len(df)):
6          distanceMatrix[i][j],_ = fastdtw(df[i],
7                           df[j], dist = euclidean)
8          distanceMatrix[j][i] = distanceMatrix[i][j]
9
10 for n_clusters in range(2,10):
11     model = KMedoids(n_clusters, distanceMatrix)
12     centers, members, costs, tot_cost, dist_mat =
13         model.fit(df, verbose= False)
14     score = silhouette_score(distanceMatrix, members,
15                                   metric="precomputed")
```

Listing 2.1: Distance Matrix and Silhouette Analysis in Python

Successively, by comparing the Silhouette scores we could find out how many clusters to create. Applying the algorithm reported in the second part of the code in 2.1, we obtain as a result the table 2.4. Since the best Silhouette score is reached for $K = 2$ and its value is close to 0.5, it means that partitioning all the 22 units in just two groups might be a right clustering.

Clusters	Silhouette score
2	0.485
3	0.085
4	0.102
5	0.322
6	0.145
7	0.158
8	0.183
9	0.169

Figure 2.4: Table reported the Silhouette score for each value of K.

Finally, the last step is to apply K-Medoids with $K = 2$ and look at the results. The output vector of size 22, as the number of vehicles, containing for each unit its cluster label is

$$[\,0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\,].$$

The output has to be interpreted as follow: the first vehicle in the unit list belongs to the group number 0, together with the second and the third; the fourth unit is labeled with 1, so that it forms a group on its own; finally, the last 18 units are in the same partition as the first three. It's quite strange that every vehicles is put inside the same cluster but the fourth one. It means that this unit has a totally different behaviour, in the sense of sequenciality of CAN values, with respect to the others. In conclusion, for the rest of the analysis we will consider, if we need to, that unit as a separate individual group, by creating for it a different mathematical model.

2.3 Diagnostic message 1 (DM1) analysis

Techniques for predictive maintenance are designed to help determine the conditions of active and correct performing vehicles in order to estimate when maintenance should be performed. This approach provides cost savings over routine or time-based preventive maintenance, because tasks are performed only when warranted. The main promise of predictive maintenance is to prevent unexpected unit errors. By knowing which part of the vehicle needs maintenance, the work can be better planned and almost all the "unplanned stops" are transformed to shorter and fewer "planned stops". Other potential advantages include increased vehicle lifetime and fewer accidents with negative impact on them.

2.3.1 What is a Diagnostic Message 1

In this work the way we understand errors is through the active diagnostic messages (SAE J1939 protocol, see [HPL02]), which are sent by the vehicles when a problem occurs. Of course the range of problems we deal with might be so wide, passing from a simple and shallow error of some not useful parts to a deep fault of the engine work. Our main issue concerns the interpretability of those DM1s. Each DM1 is completely identified by the Diagnostic Trouble Code (DTC), which includes the Suspect Parameter Number (SPN) and the Failure Mode Identifier (FMI). So, the message is codified and represented by these two parts. The first number, the SPN, is assigned to each parameter of a parameter group or component. Of course, it is used for diagnostic purpose and its function is to report and identify abnormal operation of a Controller Application (CA). The second number, the FMI, indicates why that parameter is abnormal and in which sense (too low, too high, ecc. . .). In this sense, the basic analysis has the aim to find out whether the situation connected to the DM1 has to be treated as relevant or might be skipped. Since the meaning of each DM1 is unknown for us, the decision has to come from the knowledge acquired through the data.

When a failure is captured, the related DM1 is sent by the unit device to the remote server. So one new line is written in the dataset, composed of the

timestamp, the related couple SPN-FMI which indicates in a unique way what is the considered DM1, and a few other more technical informations. This line means that in this exact timestamp the DM1 starts. You can imagine as if an indicator lamp turn on while you are driving your car. Then we have two case for the next timestamp:

- If an other DM1 (or more than one) is captured, then the dataframe will have one new line for each still active DM1. It means that if we have for example three active DM1s and then an other problem occurs, four new lines having the same timestamp but different SPN-FMI will be written in the dataframe.
- If a DM1 is solved by maintenance, then two cases more are possible:
  - if there are still active DM1s, one line per active DM1 will be written in the dataset (but the solved DM1).
  - if no other DM1s are still active, then one new line containing SPN = 0 and FMI = 0 meaning "no still working DM1s" is reported.

This convention has to be kept in mind for the next part regarding DM1 lifetime. On the other hand, once our DM1 dataset will be created, we won't use this convention anymore because for each DM1 we will have a label indicating how it disappeared and how long does it take to be solved, if it is.
The next step is related to the creation of the lifetime dataframe through an accurate DM1 analysis.

2.3.2 DM1 data exploration: co-occurrence analysis

In this section we are going to explore the dataframe containing DM1 messages and for each DM1 we will study if any other message occurs frequently in the same moment.

At the beginning, the most relevant aspect regards the numerousness of observations. In fact, as we deal with rare and sparse events, it could be normal that a lot of DM1s occur just few times in each vehicle or even that they have never appeared yet. So, for this aspect, the final choice of the right DM1 to build the

model on it has to involve DM1s with at least 30-40 observations (summing up the amount per vehicle). Then, an other symptomatic aspect is the correlation between one DM1 occurrence and the others. It means that the more DM1s occur at the same time, the deeper might be the fault of the unit. In this way, the analysis is directed to understand if couples of DM1s appear frequently together, in the sense that the number of times they are sent simultaneously respect to the number of times they appear is greater than a fixed threshold. Since we prefer a strong selection, the threshold is put at 0.75 and we consider only those DM1s which occur at least 30 times. The results are listed in the table 2.5.

	DM1_A	DM1_B	DM1_A count ts	DM1_B count ts	DM1_A, DM1_B count ts
3 DM1s	65702-2	131238-2	177	177	177
	65717-2	131252-2	89	90	89
	65736-3	131252-2	88	90	83
	65736-3	65717-2	88	89	83
6 DM1s	524016-2	523803-9	74	79	65
	523867-12	523803-9	80	79	71
	523803-9	3224-9	79	76	72
	523867-12	3224-9	80	76	72
	523939-9	3224-9	78	76	70
	523939-9	523803-9	78	79	69
	523939-9	523867-12	78	80	75
	524029-2	3224-9	77	76	71
	524029-2	523803-9	77	79	68
	524029-2	523867-12	77	80	69
	524029-2	523939-9	77	78	68
	524016-2	3224-9	74	76	60
	524016-2	523939-9	74	78	60
	524029-2	524016-2	77	74	59
	65642-2	65641-2	22	22	22

Figure 2.5: Table reported the amount of observations per couple of DM1 occurring at the same timestamp.

It illustrates some useful informations about recurrent pairs or, more in general, groups of DM1s. In fact, visualizing DM1s as points in a graph (see Figure 2.6) and the couple as the relationship, you should note that there are many loops inside the graph. So, it means more than two DM1s occur together many times. As a result, we conclude that:

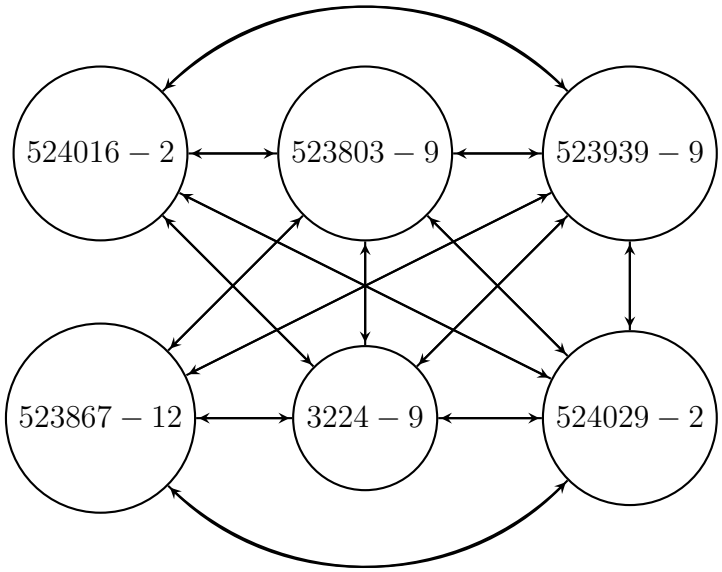


Figure 2.6: Graph showing the loop within the group of 6 DM1s. There is one exception related to nodes $524016 - 2$ and $523867 - 12$ which are not connected.

- DM1s "65702-2" and "131238-2" are broadcast by six different units and they appear almost surely together;
- There is a long cycle of DM1, and that means all the members of the couples belong to a unique group of recurrent DM1. This group is composted of: "523867-12", "523803-9", "3224-9", "524029-2", "523939-9" and "524016-2"; the limit is that only one unit is involved;
- Three different DM1s are always sent together, with respect to the threshold, by a variety of 10 units: "65717-2", "131252-2" and "65736-3".

2.3.3 DM1 data exploration: lifetime analysis

In this section we are going to figure out the lifetime of each type of DM1. Some statistics will be computed about the engine start and stop cycles, and the time distance between DM1 messages and the corresponding timestamp of the engine

off. Furthermore, we are going to light up some very relevant considerations concerning the number of not solved messages and their lifetimes. We will make a clear division between those DM1s which are going to be recurrent in the next engine on/off cycle and those which are probably going to be solved during the off period of the unit. We'll call these two category as "persistent DM1s" and "resolved DM1s".

To have a better scenario, currently we should create a dataframe where each row represents a single couple of SPN-FMI with its duration, computed as the difference between the "death" timestamp (when it's resolved) and the "birth" timestamp (when it appears). In order to achieve that task, it seems to be necessary to look for a proper definition of "end" for each DM1. So, the question is: how can we compute the ending timestamp for each DM1 message? After careful considerations, we have summed up that all the DM1s end within three different cases. Then, for each one of these cases, we put as the ending timestamp

- the next timestamp, if in the following timestamp the same DM1 is not sent anymore by the unit;
- the next timestamp, if in the following timestamp just one DM1 is sent and it contains SPN = 0 and FMI = 0;
- the engine off timestamp, if the engine is turning off before one of the previous two situation happens.

So, in the first case it's sufficient to use the next timestamp as the end of the DM1, whereas in the second case we'll take the "0-0" timestamp and in the last case the engine off timestamp. Then computing the lifetime of each DM1 is very easy (Figure 2.7).

After having constructed the dataframe reporting for each DM1 message its lifetime, we should filter it in order not to consider multiple times the same DM1: every time that one new DM1 appears, it will be reported together with the entire list of active DM1s. So, if a preceding active message persists, a new row will be printed inside the dataframe even though the observation refers to the same DM1 as the previous row. Hence, engine on/off cycle by engine on/off cycle we

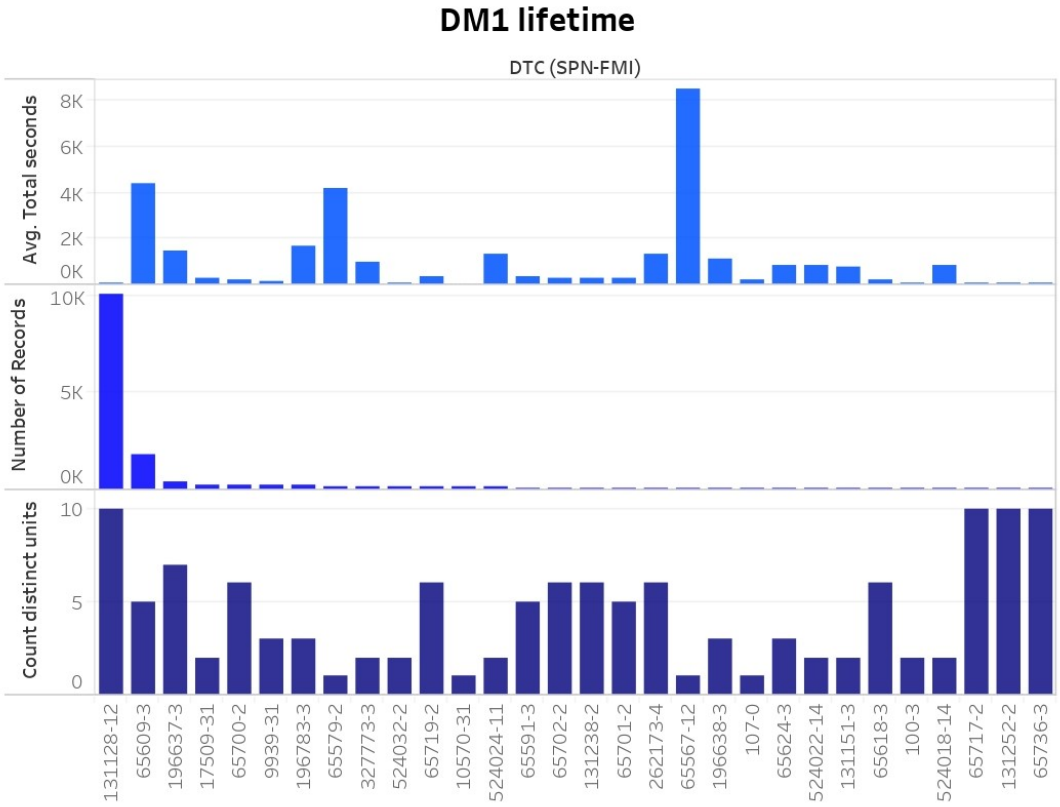


Figure 2.7: For each TDC (couple of SPN and FMI) the graph shows its lifetime, together with the number of records and how many units broadcast that DM1.

have to select only those DM1s which show the maximum total seconds value or the minimum activation timestamp. Obviously, we must group data by the engine off timestamp and, of course, by DM1. Finally, it remains much less rows in our dataframe and it is ready to be examined.

Currently we are ignoring if a DM1, emerged during one engine on/off cycle and not solved before turning off the unit, is going to persist during the next cycle. Could the vehicle be repaired during the engine off period? To answer this question we should introduce two new support dataframes to make the analysis. It's sure that, in case of positive response to the question above, we'll obtain a situation where the next on/off cycle will not contain that DM1 because it should be disappeared during the time gap between the previous engine off and the next engine on timestamp. So, the first partition is between resolved DM1 and persistent DM1. A DM1 is labelled as *resolved* if it occurs during an engine on/off cycle, it doesn't disappear before the engine is turned off and it doesn't occur again in the following engine on/off cycle. To be precise, we'd like to say that the following engine on/off is not the exact next one in time but we look for the first "long" cycle in the next 12 hours. In order to avoid to consider simple and fast trials of turning on/off the engine, where even if there was a DM1 it couldn't have enough time to occur again, we search for the first cycle of at least 30 minutes in the following 12 hours and use it as parameter to understand if the DM1 is solved for real. If there aren't long cycle, then we use the real next one. Then, it's quit natural to label those DM1 which don't belong to the solved group, but at the same time they're continuing to be active after the engine turning off as *persistent*. It means that we can't say that a real maintenance is done in the meanwhile and so, we suppose that the related problem persists over time. Finally we make an other division among the DM1s in the resolved group, based on the way they are solved. Summing up, a DM1 is labelled as:

- *Resolved*, if it's not going to appear again in the next timestamp;
- *Resolved off*, if it's active before turning the engine off and in the first 30 minutes of the next engine on/off cycle it will not be captured again;

- *Resolved by* 0-0, if in the next timestamp the message contains SPN = 0, FMI = 0;
- *Persistent*, if it's active before turning the engine off and in the first 30 minutes of the next engine on/off cycle it will be reported again.

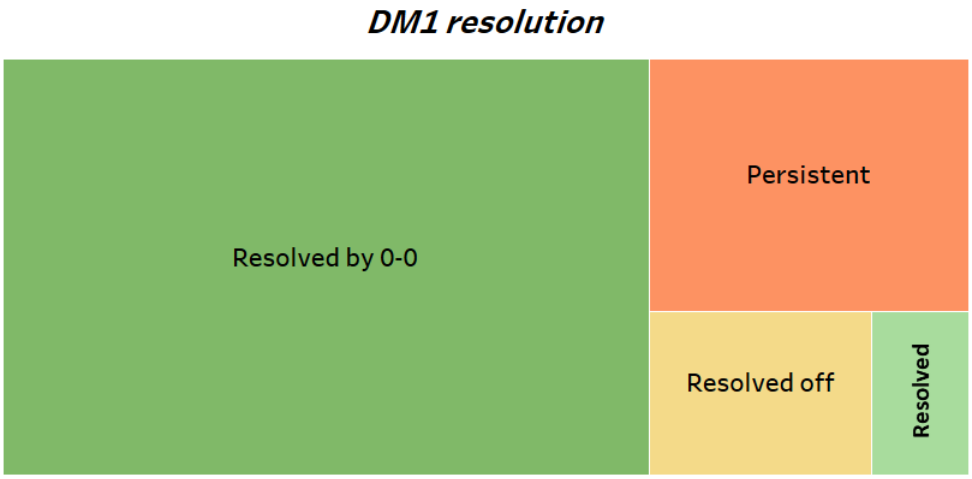


Figure 2.8: Count of DM1s class by class based on the ending reason.

In figure 2.8 it is shown the amount of DM1s per category, based on their ending. Since most of the times one DM1 per period occurs, the most crowd class is the resolved 0-0 one. In fact whenever a DM1 is resolved, if it is the only active one, there appear a line with DTC 0-0.

After an accurate analysis of the results and a comparison with a domain expert, the choice of which is the most worthable DM1 has to be taken. Since we found out of two very interesting cases of relevant DM1s, we decided to use both them separately.

The first choice is strictly related to the future results. Thanks to a domain expert of Storti S.p.A., we have identified a DM1 related to the engine oil pressure and in the CAN dataset it is present the strictly connected parameter about the same measure. In this case we hope for a strong dependence between the parameter and the DM1. It is the DM1 9939-31 which corresponds to the *low engine oil pressure*.

The second choice, according to the table 2.5, regards those three different DM1s that occur always together. It is a trial to see if an unknown DM1 could be predicted by using all or someone of the available SPNs. Two reasons make us opt for the choice of these 3 DM1s: for first, as we have already said, three distinct problem may mean a more influent fault of the vehicle; secondly, there are involved 10 units and this amount is substantially relevant as we deal with more various sequences (depending on the unit).

2.4 Creation of the predictive dataset and data cleaning

Selected the worthable DM1, we have to compose together both the two dataset containing CAN messages and DM1 messages. The creation of the right dataset is very essential to reach the prediction task, since the response variable should be expressed as well as possible. We add two columns in the dataframe of CAN messages. The first one is called "DM1" and the elements are booleans. A true value means that the line, which contains the received CAN message together with the two timestamps where it has been computed, has a non empty intersection with the interval of the DM1 lifetime. The false, instead, indicates that it is a message in which no problem has arisen. The second column, called "next_dm1_ts_on" contains the first next timestamp in which the DM1 appears. It is particularly useful to know what's the temporal distance between this CAN parameter and the next time the DM1 appears.

Then, the next step concerns how to arrange the rows and columns of the dataframe. A natural choice is to create a new column for each SPN using the timestamps of the CAN messages to identify each row. In this way the new dataframe will have about from 29 to 40 columns, of which the first 8 columns are from the original dataset and concern unit name and timestamps, whereas the others represents the SPNs involved in the analysis.

The following part has an important role and is about the missing data analysis. It's important to look at eventual missing data, reported in our structure as "NaN"

values. In a theoretic system, in every timestamp the device should send the CAN messages all together, one for each configured SPN, in the sense that each row of our dataframe should not have "NaN" values, since (fixed a timestamp) just two cases would be admitted: or every SPN has a CAN value or no SPNs have it. However in the real world, the situation is not as clean as we would want: devices could have many problems connected with the network or the environment and, as a result, messages could have been sent in different moments. By exploring data, it's easy to realize that often a part of the CAN messages are sent during a timestamp and the other one after one or two seconds. Thankfully there is no overposition of SPNs, in the sense that it doesn't happen that a CAN is sent once and then is sent again few second later. So, the ideas is to adjust the rows joining together the rows where the interval between them is less or equal to three seconds. The rest of the rows, if any, are dropped if a "NaN" value is hidden inside it.

The last clarification concerns all the messages that don't have a DM1 in the future because it's not happened yet. Basically we have to introduce a sort of agreement regarding a treatment for those CANs messages which do not lead to a new DM1 observation. Easily, in this case a false DM1 timestamp is set as default: "2014-01-01". This data is surely in the past respect to the CAN timestamp, since the first timestamp we got is close to the end of 2014. The decision to put a data in the past is functional for the analysis we will make and it can be used for every kind of problem.

Chapter 3

Time Series

For this chapter we take cue from the article [BDC02]. Let's begin giving the simple basic definition of stochastic process.

Definition 3.1. A *stochastic process* is a parametrized collection of random variables

$$\{X_t\}_{t\in T} \tag{3.1}$$

defined on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and assuming values in $\mathbb{R}^n$.

An important part of the analysis of a time series is the selection of a suitable probability model (or class of models) for the data. To allow for the possibly unpredictable nature of future observations it is natural to suppose that each observation x_t is a realized value of a certain random variable X_t. So, a time series is a set of observations x_t, each one being recorded at a specific time t. A *discrete-time* time series, which is the type we are going to use, is one in which the set T of times at which observations are made is a discrete set, as is the case, for example, when observations are made at fixed time intervals. *Continuous-time* time series are obtained when observations are recorded continuously over some time interval, e.g., when $T = [0, 1]$.

Remark 1. We will frequently use the term time series to mean both the data and the stochastic process of which it is a realization. When we talk about the process, we will use uppercase letters while the lowercase letters stand for the data points.

Definition 3.2. Let $\{X_t\}$ be a time series with $\mathbb{E}(X_t^2) < \infty$. The *mean function* of $\{X_t\}$ is

$$\mu_X(t) \coloneqq \mathbb{E}(X_t) \quad \forall\, t \in \mathbb{R}. \tag{3.2}$$

The *covariance function* of $\{X_t\}$ is

$$\gamma_X(r,s) \coloneqq \mathrm{Cov}(X_r, X_s) = \mathbb{E}\left[(X_r - \mu_X(r))\,(X_s - \mu_X(s))\right] \quad \forall\, r,s \in \mathbb{R}. \tag{3.3}$$

Definition 3.3. Let $\{X_t\}$ be a time series and let $F_X(x_{t_1+h}, \ldots, x_{t_n+h})$ represent the cumulative distribution function of the unconditional (that is, with no reference to any particular starting value) joint distribution of $\{X_t\}$ at times $t_1+h, \ldots, t_n+h$. Then, $\{X_t\}$ is said to be strictly stationary if

$$F_X(x_{t_1+h}, \ldots, x_{t_n+h}) = F_X(x_{t_1}, \ldots, x_{t_n}) \quad \forall\, h, t_1, \ldots, t_n \in \mathbb{R}, \forall\, n \in \mathbb{N} \tag{3.4}$$

Remark 2. Strict stationarity of a time series of kind $\{X_t,\ t = 0, \pm 1, \ldots\}$ is defined by the condition that $(X_1, \ldots, X_n)$ and $(X_{1+h}, \ldots, X_{n+h})$ have the same joint distributions for all integers h and $n > 0$.

Let's now introduce a weak concept of stationarity:

Definition 3.4. A time series $\{X_t\}$ is *(weakly) stationary* if

- $\mu_X(t)$ is independent of t;
- $\gamma_X(t+h, t)$ is independent of t for each h.

It is easy to check that if $\{X_t\}$ is strictly stationary and $\mathbb{E}(X_t^2) < \infty$ for all t, then $\{X_t\}$ is also weakly stationary. Whenever we use the term stationary we shall mean weakly stationary as in Definition 3.4, unless we specifically indicate otherwise.

Definition 3.5. Let $\{X_t\}$ be a stationary time series. The *autocovariance function* (ACVF) of $\{X_t\}$ at lag h is

$$\gamma_X(h) \coloneqq \gamma_X(h,0) = \gamma_X(t+h,t) = \mathrm{Cov}(X_{t+h}, X_t) \quad \forall\, t \in \mathbb{R}. \tag{3.5}$$

The autocorrelation function (ACF) of $\{X_t\}$ at lag h is

$$\rho_X(h) \coloneqq \frac{\gamma_X(h)}{\gamma_X(0)} = \mathrm{Cor}(X_{t+h}, X_t) \quad \forall\, t \in \mathbb{R}. \tag{3.6}$$

3.1 Sample Autocorrelation Function

As in most other analyses, in time series it is assumed that the data consist of a systematic pattern (usually a set of identifiable components) and random noise (error) which usually makes the pattern difficult to identify. Most time series analysis techniques involve some form of filtering out noise in order to make the pattern more significant.

Although we have just seen how it is defined the autocorrelation function for time series, in practical problems we start with observed data $\{x_1, x_2, \ldots, x_n\}$. To assess the kind of dependence in the data and to choose the right model for the data, one of the most important tools we could use is the sample autocorrelation function (sample ACF). If we believe that the data are realized values of a stationary time series $\{X_t\}$, then the sample ACF might be an accurate estimate of the ACF of $\{X_t\}$. The following definitions are natural consequence of the definitions for the autocovariance and autocorrelation functions given earlier for stationary time series.

Definition 3.6. Let $x_1, \ldots, x_n$ be observations of a time series. The *sample mean* of $x_1, \ldots, x_n$ is

$$\bar{x} \coloneqq \frac{1}{n} \sum_{t=1}^{n} x_t. \tag{3.7}$$

The *sample autocovariance function* is

$$\hat{\gamma}(h) \coloneqq \frac{1}{n} \sum_{t=1}^{n-|h|} (x_{t+|h|} - \bar{x})(x_t - \bar{x}), \quad -n < h < n. \tag{3.8}$$

The *sample autocorrelation function* is

$$\hat{\rho}(h) \coloneqq \frac{\hat{\gamma}(h)}{\hat{\gamma}(0)}, \quad -n < h < n. \tag{3.9}$$

Remark 3. The sample autocovariance and autocorrelation functions defined above can be computed for any data set $\{x_1, \ldots, x_n\}$ and are not restricted to observations from a stationary time series. For data containing a trend, $|\hat{\rho}(h)|$ will exhibit slow decay as h increases. On the other hand, for data with a substantial deterministic periodic component, the value $|\hat{\rho}(h)|$ will exhibit similar behaviour with the same periodicity. Thus $\hat{\rho}(\cdot)$ can be useful as an indicator of nonstationarity.

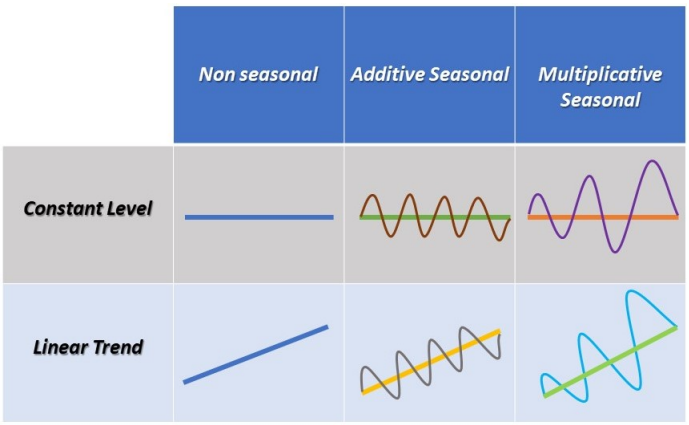


Figure 3.1: Examples of trend and seasonality for time series

A lot of time series patterns can be described in terms of three basic classes of components: trend, seasonality and noise. Since the noise is a stationary time series, let's focus on the first two concepts. As it is shown in the example in Figure 3.1, the first one represents a general systematic linear or (most often) nonlinear component that changes over time and does not repeat within the time range captured by our data. It means that trend is the component of increasing or decreasing of the time series. The second one repeats itself in systematic intervals over time. It is the component of the time series which corresponds to the pattern of the trajectory.

Summing up, through the inspection of a graph of the time series, we could represent the data as a realization of the process called "the classical decomposition model"

$$X_t = m_t + s_t + Y_t, \tag{3.10}$$

where m_t is a slowly changing function representing the *trend component*, s_t is a

function with known period d referred to the *seasonal component* and Y_t is the random noise component.

Usually the aim is to estimate and extract the deterministic components m_t and s_t in the hope that the residual or noise component Y_t will turn out to be a stationary time series. In order to do it, the following two subsections are very explanatory of what should be done.

3.1.1 Trend Analysis

There are no proven "automatic" techniques to identify trend components in the time series data; however, as long as the trend is monotonous (consistently increasing or decreasing) that part of data analysis is typically not very difficult. If the time series data contain considerable error, then the first step in the process of trend identification is smoothing.

Smoothing always involves some form of local averaging of data such that the non systematic components of individual observations cancel each other out. The most common technique is *moving average smoothing* which replaces each element of the series by either the simple or weighted average of n surrounding elements, where n is the width of the smoothing "window". Basically, it permits us to reduce the amount of noise present in the time series and to focus on the real trend. Medians can be an alternative to means. The main advantage of median as compared to moving average smoothing is that its results are less biased by outliers (within the smoothing window). Thus, if there are outliers in the data, median smoothing in general produces smoother or at least more "reliable" curves than moving average based on the same window width. The main disadvantage of median smoothing is that in the absence of clear outliers it may produce more "wavy" curves than moving average and it does not allow for weighting.

Moving average is essentially non-parametric methods for trend (or signal) estimation and not for model building. Let's assume that the time series has no seasonal components. Then, the model becomes the following:

$$X_t = m_t + Y_t, \qquad t = 1, \ldots, n \tag{3.11}$$

with $\mathbb{E}(Y_t) = 0$.

Now, let q be a non-negative integer and consider the two-sided moving average

$$W_t = \frac{1}{2q+1} \sum_{j=-q}^{q} X_{t-j} \tag{3.12}$$

of the process $\{X_t\}$ defined above (3.11). Then, for $q+1 \leq t \leq n-q$,

$$W_t = \frac{1}{2q+1} \sum_{j=-q}^{q} m_{t-j} + \frac{1}{2q+1} \sum_{j=-q}^{q} Y_{t-j} \approx m_t, \tag{3.13}$$

assuming that m_t is approximately linear in the interval $[t-q, t+q]$ and that the average of the error component is sufficiently close to zero. So, as a result, the moving average makes us achieve the goal to estimate the trend component,

$$\hat{m}_t = \frac{1}{2q+1} \sum_{j=-q}^{q} X_{t-j}, \quad q+1 \leq t \leq n-q. \tag{3.14}$$

The choice of the interval for t values is obliged by the fact that we don't know what's the value of X_t outside the interval $[0, n]$.

In the relatively less common cases (in time series data), when the measurement error is very large, other methods should be used. In fact there are many methods which filter out the noise and convert the data into a smooth curve that is relatively unbiased by outliers. One example could be the "Trend Elimination by Differencing" which consists to eliminate the trend by differencing and then to find an appropriate stationary model for the differenced series. Since it is not so useful for our problem, that doesn't concern the time series value predictions, we won't go deeper inside that aspect.

3.1.2 Analysis of seasonality

Seasonal dependency (seasonality) is the other general component of the time series pattern. It is formally defined as correlational dependency of order k between each i-th element of the series and the $(i-k)$-th element and measured by the

autocorrelation. The autocorrelation is the measure of a correlation between the two terms. Usually the value of k is called the lag. If the measurement error is not too large, seasonality can be visually identified in the series as a pattern that repeats every k elements.

A way to examine seasonal patterns of time series is the correlograms. The correlogram (autocorrelogram) displays in a graph and numerically the autocorrelation function (ACF). It means that in the graph are printed the correlation coefficients and their standard errors for consecutive lags in a specified range. Ranges of the standard errors for each lag are usually marked in correlograms but typically the size of autocorrelation is more interesting than its reliability because we are usually interested only in very strong, and so highly significant, autocorrelations.

While examining correlograms, we should keep in mind that autocorrelations for consecutive lags are generally dependent. Let's make a simple example. If the first element is closely related to the second one, and the second one to the third one, then the first element must also be somewhat related to the third, etc. Obviously what could change is the strength of the correlation. In fact it is true what we said before, but we have to remind even that weak correlations are not interesting for the problem. So, from this relation it follows that the pattern of serial dependencies can change considerably after removing the first order autocorrelation, that means after differencing the series with a lag of 1.

3.2 SPN by SPN autocorrelation analysis

Since we have a multidimensional problem, let's first give the right definition.

Definition 3.7. A p-dimensional multivariate *time series* of length n

$$X = \{X(t_l) \in \mathbb{R}^p;\ l = 1, \ldots, n\}$$

is a sequence of data points such that:

- $t_l < t_{l'}$ for $l < l'$;
- $t_l \in T = [a, b]$, $\forall\, l = 1, \ldots, n$;

- $X(t) \in \mathbb{R}^p$, $\forall\, t \in T$.

In particular, each dimension corresponds to one SPN. Actually, we are going to study only few SPNs, because looking at all the dataframe, it seems that mostly of those sequences have a similar behaviour in term of autocorrelation and trend. So, behaviours can be grouped in four families represented by SPN 100, 247, 30066, 30104, as reported in Figure 3.2 where the 4 graphs are shown.

The first step in the analysis of any time series is to plot the data. If there are any apparent discontinuities in the series, such as a sudden change of level, it may be advisable to analyze the series by first breaking it into homogeneous segments. If there are outlying observations, they should be studied carefully to check whether there is any justification for discarding them (as for example if an observation has been incorrectly recorded). However, this part of data cleaning has already been done in the previous chapter, so that all the points present in the graph have a reason to be inside the dataframe.

The first type of group is represented by SPN 100. It doesn't have any sort of trend or seasonality. However, it's relevant that there are two different zones of values: most of values fall inside the gaps $[250, 550]$ and a smaller group inside $[100, 250]$. This situation is very significant for the problem we are dealing with because vehicles perceive two states: a unit can have a normal parameter behaviour when its data fall into a normal interval, like in this case the most crowed of the two, and it may have a less dense zone meaning a sort of anomaly values. The choice of SPN 100 is not a random choice, since it is the parameter connected with the DM1 we are going to use for the model.

About the second time series, it looks like to have a very smooth and linear trend. This is motivated by the nature of that parameter: it represents the engine hours, in the sense that it measures the amount of hours when the vehicle is active. It is an incremental value, so that the correlation must be very strong. So, it's logical that its trend is increasing with the passage of time and, of course, between intervals of the same time length the increment results to be equal. In this case the trend could be eliminated just using the "Trend Elimination by Differencing" which is basically the use of the difference with the previous value instead of the

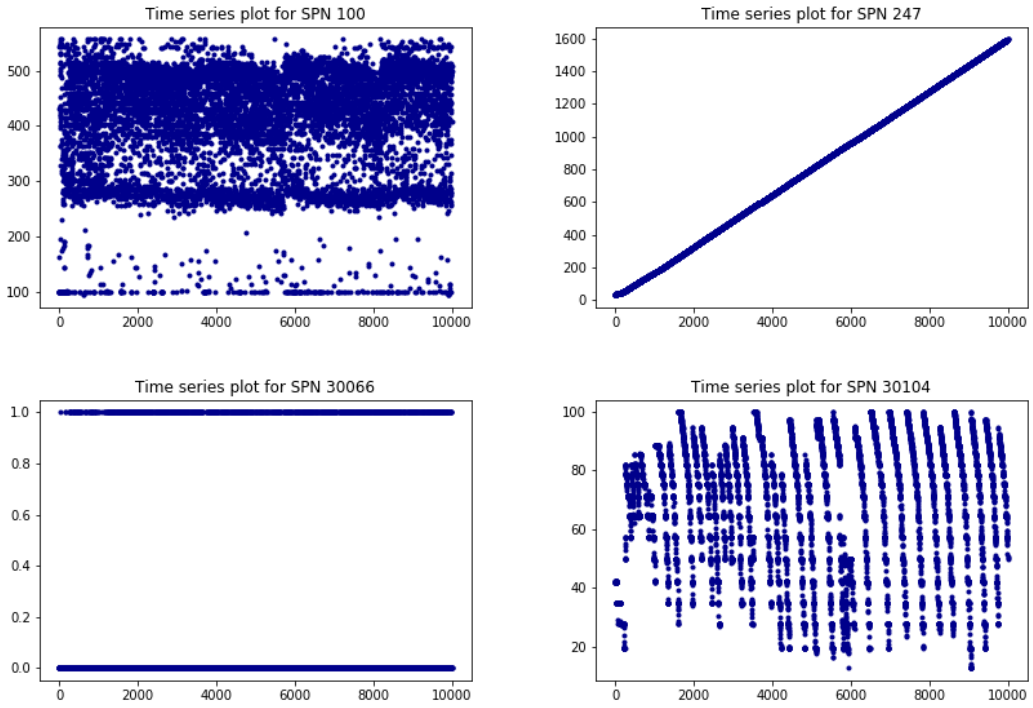


Figure 3.2: Time series plot using the first 10.000 values for 4 different representative SPNs: 100, 247, 30066, 30104.

value itself. But, since it is a non significant feature, we prefer to delete it rather than falling trivially in overfitting.

The third group contains a categorical variable taking the values of 0 and 1. Usually is too hard to identify trends or seasonalities within binary variables, unless the categories represent very different situations. So, it means that we can skip this part and ignore eventual weak patterns.

The last group is represented by SPN 30104 which is related to the level of Urea into the tank. From the graph it is immediate to recognize a seasonality inside the data: we have a lot of cycles where the SPN begins at a high level and finishes close to 20 or 30. The interpretation is quite simple: the first observation is taken when the tank is full and then, going on with the work, the container is gradually going to empty. So we can expect that the ACF will be with many peaks and maybe not so high for long lags.

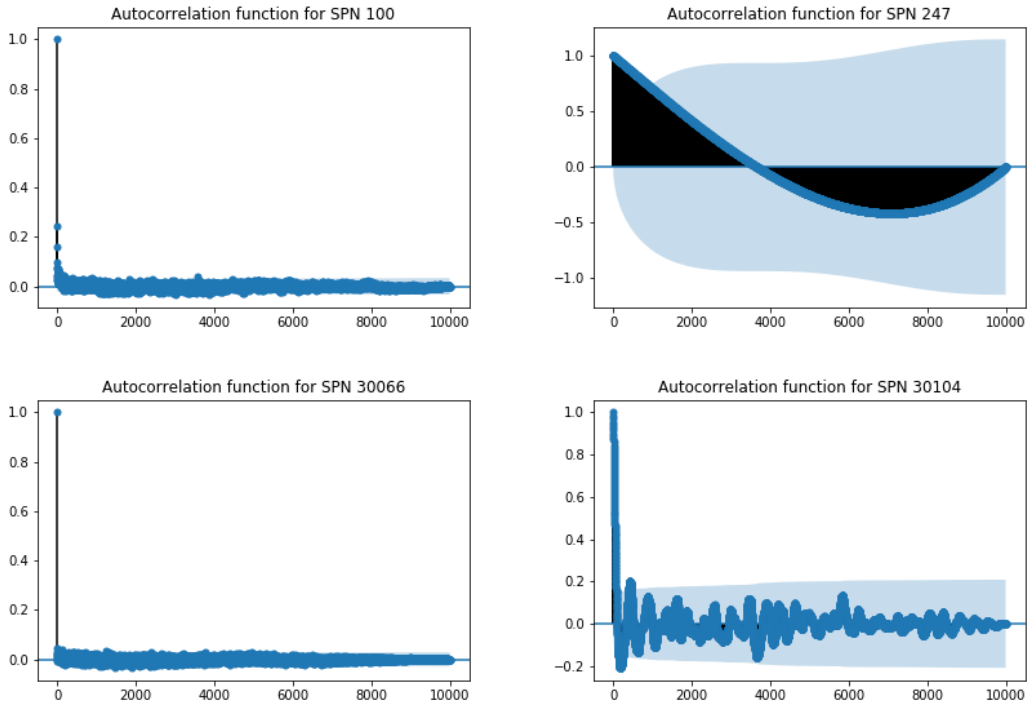


Figure 3.3: Autocorrelation function using the first 10.000 values for 4 different representative SPNs: 100, 247, 30066, 30104.

In the Figure 3.3 it is shown for each representative SPN its autocorrelation function, taking only the first 10.000 observation of one unit as significant sample. The first and third SPN have very low autocorrelation, meaning that the time series don't have long strong correlations. The second one, as we said before, has a very interesting pattern that confirms us the trivial dependence among the consecutive values. Even the last SPN has a notable autocorrelation which persists even for long lags. On the other hand, ACF for SPN 30104 takes values within a gap of $[-0.2, 0.2]$ which is not so high as we could have intended.

This chapter has to be considered as a preliminary analysis for the future creation of the model. It is useful just to figure out the general behaviour of time series we are going to use. The informations we take out of it are the following:

- SPN 247 has a trivial trend and it doesn't add informations, so it can be ignored.

- We have a group of SPN with a strong seasonality, like SPN 30104, which is due to the cycle of a unit work.
- Quantitative variables have no strong trends of seasonalities.
- There is a group of SPNs represented by SPN 100 with two or more dense zones of values. There could be possible to detect anomalies inside them looking for values in one of those intervals.
- Many SPN parameters have low autocorrelation with wide lags.

In the next chapter we are going to select the interesting SPNs, which are useful for real into the analysis and for the model we will build.

Chapter 4

Feature Selection for High-Dimensional Data

4.1 Introduction to Feature Selection

Feature selection has the role to identify a subset of original features from a given dataset while removing irrelevant and/or redundant features. It has the tasks to improve the performance of the predictors, to provide faster and more efficient predictors, and to provide a better knowledge of the underlying process that generated data. In order to have a better model, we need to include high predictive capability features and exclude low ones from the original group. For this reason, the methods choose a subset of the original features to be used for the entire predictive work. Hence, only the data generated from those selected features need to be collected in future, ignoring all the other ones which are not important for the analysis. Time series data contains temporal ordering, which makes its feature selection use different methods from the normal approach. A time series is called a *multivariate time series* when the number of features is equal to, or greater than 2. A multivariate time series is naturally arranged in an $n \times m$ matrix, where n is the number of observations and m is the number of variables, for example in our case, CAN parameters.

Data cleaning is an important part of the DM1 prediction process in order to

eliminate useless variables, such as CAN parameters that have nothing to do with the kind of error we would want to predict. Since most of the variables are unknown, as most of the diagnostic messages, discovering a new significant subset of feature might be important even for practical interpretability reasons. For example, if the method selects only m_0 variables concerning the engine oil, it lead us to think that the chosen DM1 might be related to the engine oil.

For this part it is essential to make a net division between CAN messages close to the DM1 occurrences and CAN messages far from them, as it is done in the article [Pry14]. In fact, we want to figure out if there is a, probably weak, dependence between the first group of CAN parameters (close to DM1s) and the second one (far from them). In order to do it, we put a label to each CAN message adding a column into our dataframe. The label identifies the nature of the CAN: fixed a time gap, for example from the DM1 occurrence until 30 hours before it happens (going backward), all the rows of our dataframe, falling into this time gap, are labeled as "Faulty". It means that they are changing because of the use of the vehicle and they are leading the unit towards a DM1 signal. The remaining messages are labeled as "Normal", since they are far from the DM1 and so they represent a normal behaviour of the unit. Comparing these two groups for each chosen time label, we should figure the potential dependence out. In practice, we add columns into the dataset called "$x.0h$", where x is an integer multiple of 3 from 3 to 48 (16 new columns).

Before exploring more accurately all the methods for feature selection, let's have a look at the Figure 4.1. In the violin plot a subset of features corresponding to all the quantitative (real) SPNs is shown. Violin plots are similar to box plots, except that they also show the probability density of the data at different values, smoothed by a kernel density estimator. At the first sight we immediately see in graph 4.1 that only one feature really changes its distribution between the two classes, while the others may have just a little, or maybe even not significant, mutation. So, as a result, we could expect that the subset of feature will contain only SPN 100.

In the next sections we are going to discuss three different canonical methods

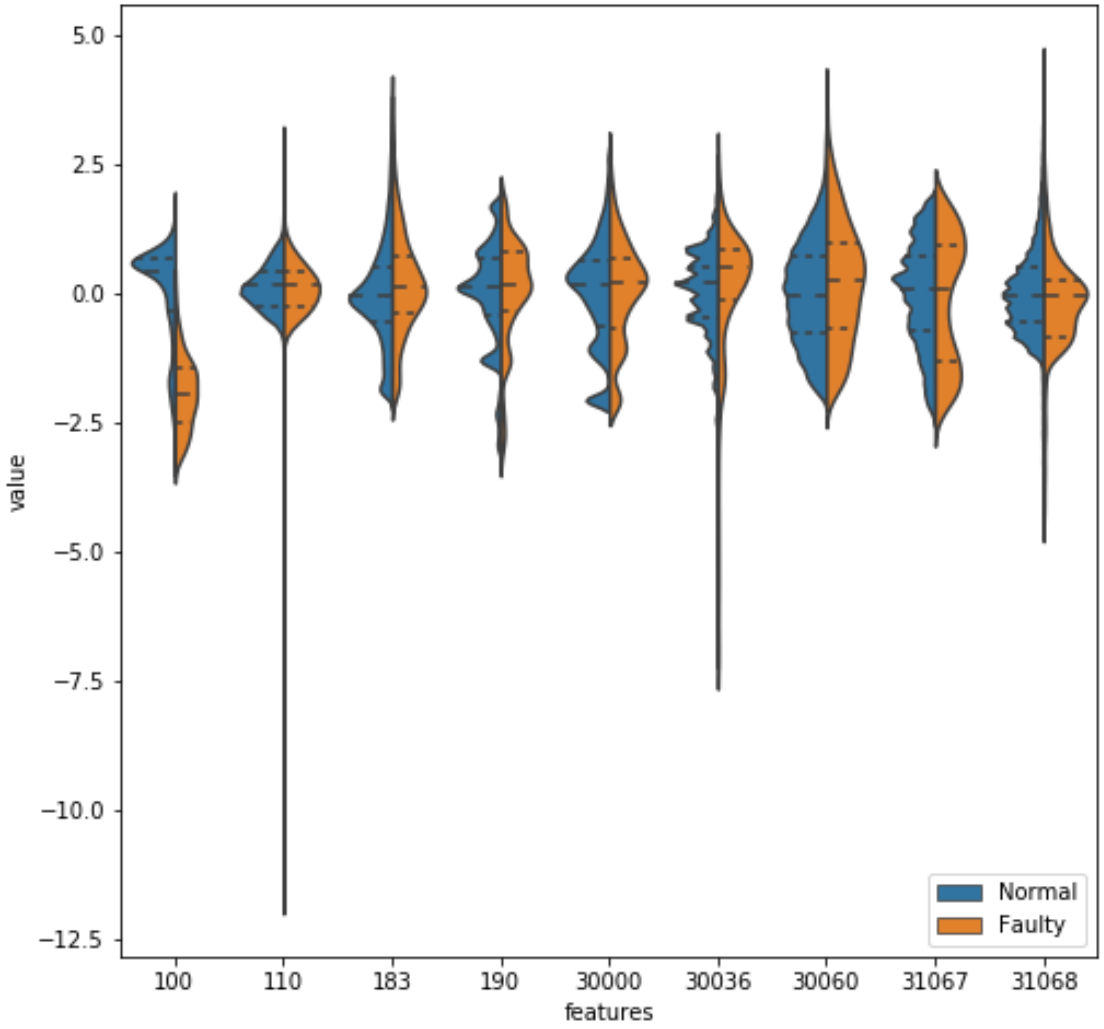


Figure 4.1: Violin plot representing for each feature its distribution inside the two classes.

for feature selection and one more uncommon, based on the DTW distance.

4.2 Entropy and Mutual Information

The concept of "information" is too wide to be summarised completely by a unique, brief and precise definition. However, fixed any probability distribution, we define for it a quantity called *entropy*, which has many properties that agree with the intuitive common notion of measure of information, in the sense that it corresponds to the general idea we should have. This notion is then extended to define *mutual information*, which is a measure of the amount of information that a random variable X contains about another one Y. Then we can see entropy as an estimate of the self-information of a random variable. Finally, mutual information is a special case of a more general quantity called *relative entropy*, which is defined as a measure of the distance between two probability distributions [CT91].

4.2.1 Entropy

Firstly let's define the concept of entropy, which is a measure of uncertainty of a random variable.

Definition 4.1. Let X be a discrete random variable defined on an alphabet $\mathcal{X}$, with probability distribution function $p_X(x) = \mathbb{P}\{X = x\}$ for any $x \in \mathcal{X}$. The *entropy* $H(X)$ of X is defined as

$$H(X) \coloneqq -\sum_{x \in \mathcal{X}} p_X(x) \log p_X(x), \tag{4.1}$$

with the assumption that $0 \log 0 = 0$ (by continuity).

So, the last assumption is clearly important because it means that adding zero probability terms don't change the total entropy.

Remark 4. Observe that entropy is a functional of the distribution of X, and it doesn't depend on the proper values of X, but only on its probabilities.

Now, we derive the immediate consequences of the definition. For any random variable X,

$$H(X) \geq 0. \tag{4.2}$$

In fact, from $0 \leq p_X(x) \leq 1$, it follows that $-\log p_X(x) \geq 0$ and so the inequality above.

Remark 5. Entropy might changed from one base to another one by multiplying by the appropriate factor.

To prove it, assume that $H(X)$ is related to the log base a. Then

$$H_b(X) = (\log_b a)\, H_a(X), \tag{4.3}$$

since $\log_b p = (\log_b a) \cdot (\log_a p)$.

4.2.2 Joint Entropy and Conditional Entropy

In information theory, the conditional entropy quantifies the amount of information needed to describe the outcome of a random variable Y given that the value of another random variable X is known. Before talking about conditional entropy, we have to extend the definition of entropy to more variables.

Definition 4.2. Let X and Y be two discrete random variables on two alphabets $\mathcal{X}, \mathcal{Y}$ and let $p_X(x)$, $p_Y(y)$ be the respective probability density functions. The *joint entropy* $H(X,Y)$ of the pair (X,Y) with a joint distribution $p_{X,Y}(x,y)$ is defined as

$$H(X,Y) \coloneqq -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_{X,Y}(x,y) \log p_{X,Y}(x,y). \tag{4.4}$$

For the sake of simplicity, when the notation is not misleading, we indicate each distribution with $p_{\dots}$ instead of $p_{\dots}(\dots)$.
Then we can also define the conditional entropy of Y given X as the expectation of the entropy of the conditional distribution $(p_{Y|X})$, averaged over the conditioning random variable X.

Definition 4.3. Let X and Y be two discrete random variables on two alphabets $\mathcal{X}, \mathcal{Y}$ and $(X, Y) \sim p_{X,Y}$, $Y|X \sim p_{Y|X}$. Then the *conditional entropy* $H(Y|X)$ is defined as

$$H(Y|X) \coloneqq \sum_{x \in \mathcal{X}} p_X H(Y|X = x) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_{X,Y} \log p_{Y|X}. \tag{4.5}$$

Remark 6. Note that $H(Y|X) \neq H(X|Y)$. However,

$$H(X) - H(X|Y) = H(Y) - H(Y|X), \tag{4.6}$$

is a property that we will discuss and prove later.

The conditional entropy and the joint entropy are connected through the chain rule formula.

Theorem 4.1 (Chain rule)**.** *Let X, Y be as in definition 4.3. Then,*

$$H(X, Y) = H(X) + H(Y|X). \tag{4.7}$$

4.2.3 Relative Entropy and Mutual Information

The relative entropy is a measure of the distance between two distributions. It is represented by the symbol $D(p||q)$ and it is a measure of the inefficiency obtained by assuming that the true distribution is q when the real true distribution is p. For example, if we knew the true distribution of the random variable, then we could build an approximate distribution using a mathematical technique, and finally we could measure the distance between the real and the built distributions. In this case the measure could be computed using the relative entropy as score of goodness.

Definition 4.4. Let p and q be two different approximations of the probability distribution function of X, discrete random variable. The *relative entropy* or *Kullback - Leibler* distance between p and q is defined as

$$D(p \,||\, q) \coloneqq \sum_{x \in \mathcal{X}} p(x) \cdot \log \left(\frac{p(x)}{q(x)} \right) = \mathbb{E}_p \left[\log \left(\frac{p(X)}{q(X)} \right) \right] \tag{4.8}$$

Then, we introduce the concept of mutual information. As we said before, it quantifies the "amount of information" (usually expressed in bits) obtained about one random variable by observing another one. In a certain sense, it is a measure of the reduction of the random variable uncertainty due to the knowledge of another, comparing the distribution of values. For its computation, we should compare the joint distribution of two random variable with respect to the product of both the marginal distributions. In fact, as a support of the theoretical concept above, if two random variables are independent, then the joint distribution is equal to the product of marginal distributions. So, the more is "far" the joint distribution from the product, the more one variable explains about the other. But, let's give the formal definition of mutual information.

Definition 4.5. Consider two random variables X and Y with a joint probability density function $p_{X,Y}$ and marginal probability density functions p_X and p_Y. The *mutual information* $I(X,Y)$ is the relative entropy between the joint distribution and the product distribution $p_X\, p_Y$, that is

$$I(X,Y) \coloneqq \sum_{x\,\in\,\mathcal{X}} \sum_{y\,\in\,\mathcal{Y}} p_{X,Y} \log\left(\frac{p_{X,Y}}{p_X p_Y}\right) \;= D(\, p_{X,Y} \;||\; p_X p_Y\,). \tag{4.9}$$

Mutual information can be equivalently expressed as

$$\begin{aligned} I(X,Y) &= \sum_{x,y} p_{X,Y} \log\left(\frac{p_{X,Y}}{p_X p_Y}\right) \\ &= \sum_{x,y} p_{X,Y} \log\left(\frac{p_{X|Y}}{p_X}\right) \\ &= -\sum_{x,y} p_X \log p_X + \sum_{x,y} p_{X,Y} \log p_{X|Y} \\ &= -\sum_{x} p_X \log p_X - \left(-\sum_{x,y} p_{X,Y} \log p_{X|Y}\right) \\ &= H(X) - H(X|Y), \end{aligned} \tag{4.10}$$

where $p_{X,Y} = p_{X,Y}(x,y)$, $p_{X|Y} = p_{X|Y}(x|y)$ and $p_X = p_X(x)$. In addition, for the

last equality we used the two definitions 4.1 and 4.3. From the result in 4.10 we can see that the mutual information is the reduction of uncertainty of X due to Y. As obviously the definition of mutual information shows, since it is symmetric by changing the role of X with Y, we obtain the same value. So, this is the proof of property 4.6

$$H(X) - H(X|Y) = I(X,Y) = I(Y,X) = H(Y) - H(Y|X). \tag{4.11}$$

Another important property, which explains why we said that the entropy is a measure of uncertainty of a random variable, is the following. By using the mutual information and the property 4.10

$$I(X,X) = H(X) - H(X|X) = H(X), \tag{4.12}$$

so that the entropy is the measure of how much X explain of itself (it is referred to as *self-information*). Let's summarize the properties we treated above and someone else less important in a theorem.

Theorem 4.2 (Mutual information and entropy)**.**

$$I(X,Y) = H(X) - H(X|Y), \tag{4.13}$$

$$I(X,Y) = H(Y) - H(Y|X), \tag{4.14}$$

$$I(X,Y) = H(X) + H(Y) - H(X,Y), \tag{4.15}$$

$$I(X,X) = H(X). \tag{4.16}$$

A relevant recap of theorem 4.2 is shown in Figure 4.2, where the connections between all the concepts reported in this chapter are illustrated.

4.2.4 Mutual Information for Feature Selection

After having introduced the theoretical part about mutual information, in this section we use that concept to make a feature selection of our CAN parameters

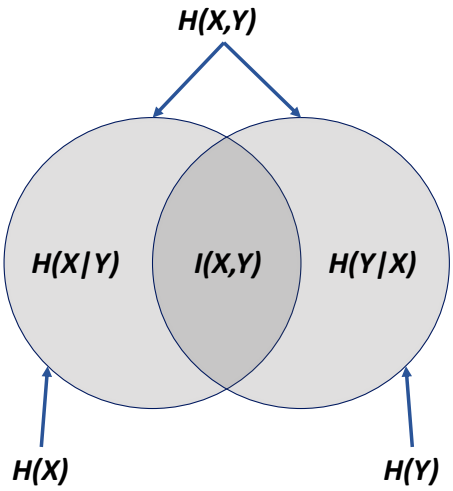


Figure 4.2: Venn diagram showing additive and subtractive relationships about information measures associated with correlated variables X and Y.

[Mur17]. First of all, we have to highlight that the response variable is the one which takes the value "Faulty" whether the timestamp of the line falls inside the fixed time gap, "Normal" otherwise. It can be seen even as a binary variable, where 1 means "Faulty" and 0 stands for "Normal".

Here we discuss the estimation of mutual information between two discrete-time time series. Note that mutual information is a deterministic function of the joint density of the underlying random processes. Therefore, it can be estimated by first estimating the (local) joint distributions, and then using these densities to compute mutual information. For this approach it is possible to use a kernel density estimator for the local densities. On the other hand, the estimation method we use to solve our problem is based on the K Nearest Neighbours (kNN) algorithm. It is important to emphasize that when estimating densities or probabilities, commonly we assume that the underlying process are stationary, ergodic, and smooth.

Let's consider two random variables X and Y, where X is one SPN at the time and Y is the response variable. Observing N pairs (X_i, Y_i) from the joint probability density function $p_{X,Y}$, we are interesting to estimate the MI using the

formula 4.15

$$I(X,Y) = H(X) + H(Y) - H(X,Y).$$

Fixed $k \in \mathbb{N}$, the distance between data point we select is the l^2-distance. Assuming a uniform local density in a small environment around each sample (this assumption implies that the density p_X is smooth), we can use the set of distances among data points to estimate the local distribution. Without falling deeper, since we assume a local homogeneous density, we can use a ball of radious equal to the kNN distance and approximate the density within all the points inside the ball. It means that we use the $k-$th distance for the approximation. To make it simpler, we don't go deeper insider the method used for the approximation, leaving in this way freedom of choice.
Then, we can estimate, by using the approximated density, the entropy via empirical averaging. So, indicating the estimation of functionals with the hat symbol ($\hat{}$), it follows that

$$\hat{I}(X,Y) = \hat{H}(X) + \hat{H}(Y) - \hat{H}(X,Y). \tag{4.17}$$

The negative aspect of this approach is that theoretically we can choose different k values for estimators, with different kNN distance. Since the estimators are not coupled, for a finite number of samples the bias can be non-negligible. This method is exceeded by a more accurate one, which is to modify the estimators of the first two individual entropy terms such that their correlation with the estimator of the joint entropy term will be higher, leading to a smaller bias.

Results

So, SPN by SPN we compute the mutual information between the X variable and the response variable Y. For the kNN distance we select $k = 1000$. The results depend on the chosen time label gap and are shown in figure 4.3.

Looking at the picture 4.3, we note that the mutual information is very low, meaning that there is a general weak dependence between SPNs and DM1 occurrences. So, because of these low values, the method we decide to apply doesn't

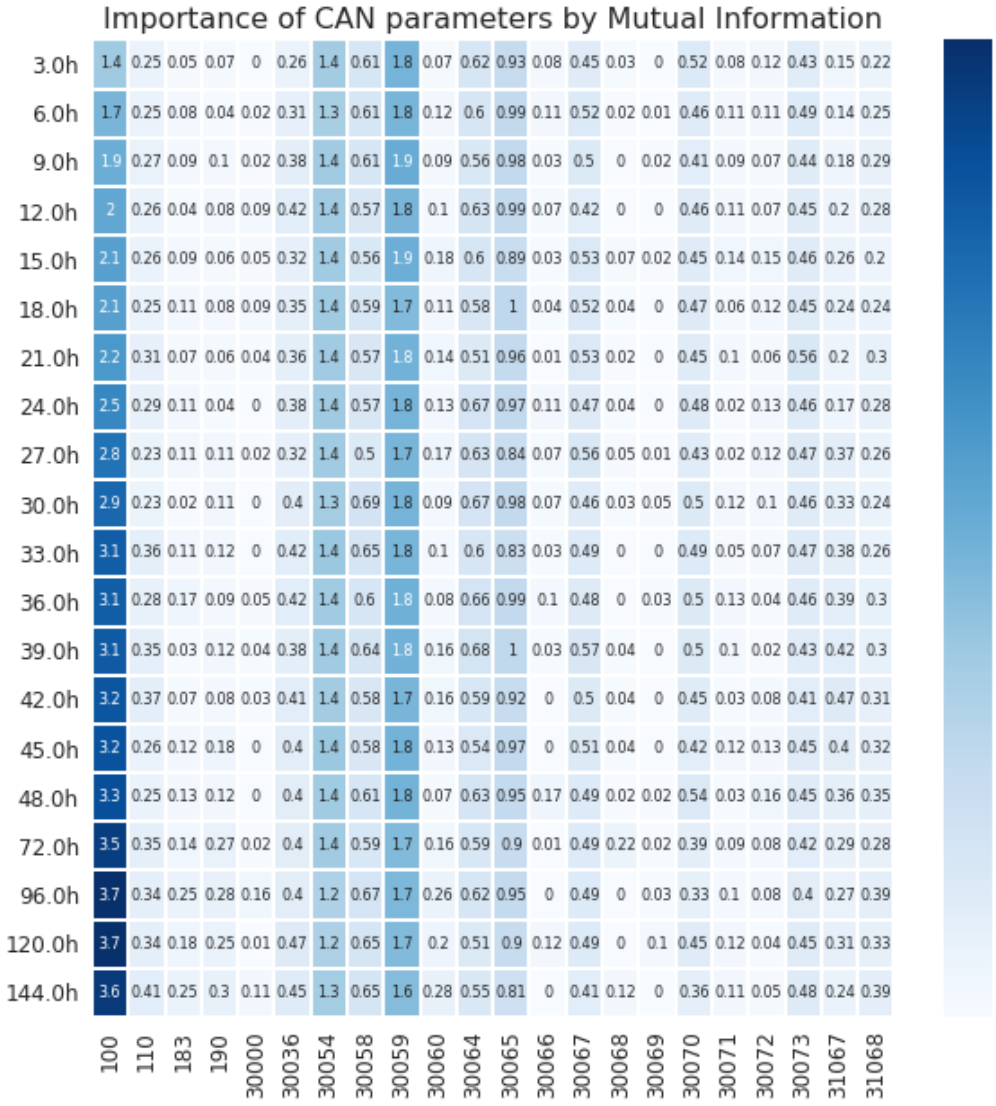


Figure 4.3: Heatmap showing for each time label gap (row) the mutual information of all the SPNs. All the mutual information values in figure are to be intended as percentage.

take high scores into account but we put a very low threshold in order not to have an empty feature set (mutual information score $> 0.5\%$). Mutual score equal to 0 means that the two variables are independent, and our results are very close to 0, so that it seems quite natural to expect low final statistics by predictive models. Even the next technique uses an absolute value to quantify the dependence between each SPN and the response variable. However, the last one considers relative scores instead of absolute one: in this way we can use a threshold more significant than before, because all the values are normalized with respect to the others.

4.3 Support Vector Machine Recursive Feature Elimination

4.3.1 Feature ranking with SVM-RFE

Support vector machine recursive feature elimination (SVM-RFE) is an embedded Feature Selection algorithm [CL11]. It uses criteria connected with the weights of the coefficients in SVM models to select features, and recursively removes features that have the smallest weight. There are two different versions of the algorithm: linear and nonlinear. The second one uses a special kernel strategy and it has to be used when the optimal decision function is obviously nonlinear. Since it is a backward elimination method, SVM-RFE has the role to model the potential dependencies between all the features. Compared to others Feature Selection methods, this algorithm has 3 peculiarities:

- It is further away from overfitting;
- It uses all the available data;
- It is much faster even if there are a lot of features (when the set of feature is very large, the gap is higher).

However,there is an issue related to SVM-RFE that makes the method not as perfect as it seems to be. If a set of the features contains variables highly correlated to each other, the defined criterion of features selection will be influenced, and their importance could be underestimated. It means that the correlated features received smaller weights due to the shared role they have in the classification models. So, instead of assessing the real contribute of each variable in the model, it penalises the entire group because of the shared relation. This is a crucial problem, especially for sensor features that are often correlated. The solution that was found is the following. When a set of features are removed in one iteration of recursive feature elimination algorithm, the group could be full of correlated features and they may be removed all together at the same time. Obviously, there might be two causes of the elimination: features could be truly irrelevant or the criterion underestimates

their importance. In both the two cases, we can insert a representative feature of the entire eliminated set back to the actual feature list in the next step. So, it can be evaluated again without the influence of all the others.

Now let's explain how the method works. First of all, we have to keep in mind that the output of the SVM-RFE is a ranked feature list, where each feature has the rank based on its importance. Then we can select the most significant feature using a top-ranked method. The way the features are ranked is totally related to the SVM algorithm, which is a common method for classification. Its intuition is to find a hyperplane which separates the data points with the largest margin as possible. Basically, given a set of training samples $\{x_i, y_i\}$ such that $x_i \in \mathbb{R}^p$ and $y \in \{-1, 1\}$, for $i = 1, \ldots, n$, the decision function of a linear SVM is

$$f(x) = w \cdot x + b, \tag{4.18}$$

where w represents the weight and it is computed through the training set.

To find the function f, the problem can be reformulated using the Lagrangian formulation, which introduces the multipliers α_i:

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j), \tag{4.19}$$

where the constraints are

$$\alpha_i \geq 0 \quad \text{and} \sum_{i=1}^{n} \alpha_i y_i = 0. \tag{4.20}$$

The α values are known as support vectors. After having found the coefficients α, we can compute the vector of weight w by

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i. \tag{4.21}$$

Finally, the method assign a rank to each feature k which is the square of the k-th element of w:

$$rank(k) = w_k^2. \tag{4.22}$$

The main meaning of the ranking is the following. Since the SVM creates the best separator hyperplane, it has one coefficient for each feature. The coefficients of the equation of the plan represents its orthogonal vector, so that

$$w = \begin{pmatrix} w_1 \\ \vdots \\ w_p \end{pmatrix} \tag{4.23}$$

is orthogonal to all the points belonging to the hyperplane. Then, by computing the inner product between w and a point, the result explains which is the class where the data is classified. So, the higher is the component w_k the more influent it is for the classification. As a result, higher w_k values means more significant features.

Iteration by iteration of the recursive feature elimination process, a linear SVM model is trained on data. At each step, the feature with the smallest ranking value is removed because it means that it has the weak effect on the dual classification problem. All the remaining features are used for building a new SVM model in the following iteration. This process is repeated until all the features are removed. After removing all the features, we have the order of elimination that leads us to a general ranking where the score 1 is assigned to the last feature removed, 2 is for the second to last and so on. Basically, the features are ordered by the order of removal. The heart of the method is the feeling that the later a feature is removed, the more important it should be.

4.3.2 SPN ranking based on cross-validated selection

Recursive Feature Elimination would require a specified number of features to keep, however it is often not known in advance how many features are necessary for the problem. To find the optimal number of features we use the cross-validation approach with RFE to score different feature subsets and select just the best scoring collection of features. As the other Feature Selection methods, we repeat it for each time label gap, since we want to find a result for each different gap. After

	feature	label_time
0	100	6.0h-12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
1	110	6.0h-12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
2	183	6.0h-12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
3	190	12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
4	30000	12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
5	30036	3.0h-6.0h-9.0h-12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
6	30054	6.0h-12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
7	30058	6.0h-12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
8	30059	12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
9	30060	12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
10	30064	6.0h-12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
11	30065	6.0h-12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
12	30066	6.0h-12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
13	30067	6.0h-12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
14	30068	6.0h-12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
15	30069	12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
16	30070	6.0h-12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
17	30071	6.0h-12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
18	30072	6.0h-12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
19	30073	6.0h-12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
20	31067	12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h
21	31068	6.0h-12.0h-15.0h-18.0h-21.0h-24.0h-27.0h-30.0h-33.0h

Table 4.1: Table showing for each feature the time labels where it has the rank 1. To make it more readable we stop the times at $33.0h$. All the remaining time labels appear in all the features.

computing the ranking for the SPNs importance for each time label, we select for each SPN the time labels where it has the rank 1. So, grouping SPN by SPN and merging all this time labels into a unique string, the results are shown in the table 4.1.

As we saw with the mutual information scores, even this method perceives a general weak importance. In fact we see that from time label $12.0h$ all the features are ranked as 1. This is due to the fact that the method isn't able to distinguish the variables by their importance and so it classifies all of them as the most important features. This two approach that we have seen until now look for the absolute importance of each feature within the problem. Since the relation is too weak, the next method find out the relative importance of each feature, so that we can have at least a real normalized rank.

4.4 Feature Importance by Extremely Randomized Trees

We start explaining how the method called Extremely Randomized Trees works and then we will apply it into our problem of feature selection. It basically consists on randomizing strongly both attribute and cut-point choices while splitting a tree node. In the extreme case, the method randomly picks a single attribute and cut-point at each node, and hence builds totally randomized trees whose structures are independent of the target variable values of the learning sample. In Figure 4.4 it is shown an example of tree with maximum depth equal to 3.

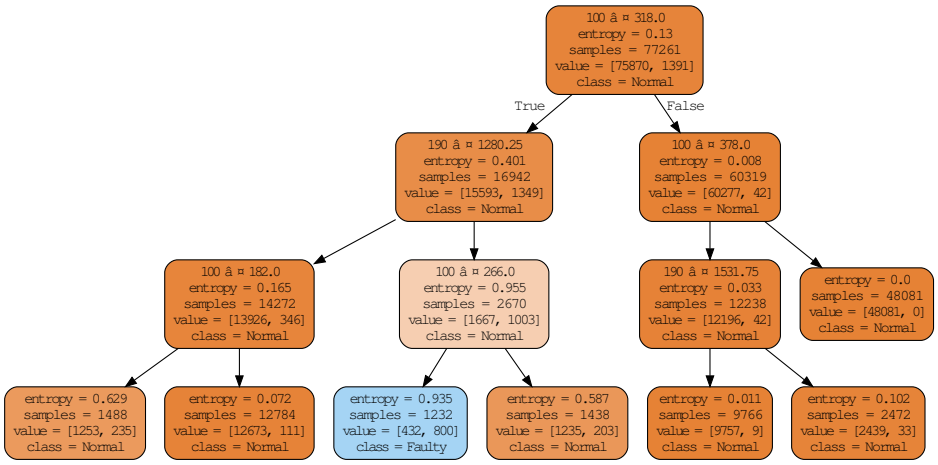


Figure 4.4: Example of tree created using all the data and setting the maximum depth to 3. Blue color represents the faulty classes, whereas the orange stands for normals.

4.4.1 Extremely Randomized Trees

The Extremely Randomized Trees algorithm (ERT) creates a set of unpruned decision trees using the classical tree based approach [YZ15; GEW06]. There are two main differences with the other methods:

- ERT splits nodes by choosing all the cut-points completely randomly.
- It uses the entire sample to grow the three and not just a part of it.

This algorithm has 2 parameters:

- K, the number of features randomly selected at each node. It determines the strength of the feature selection process.

- $N_{\min}$, the minimum number of feature for splitting a node. It determines the strength of averaging output noise

In addition, let's denote by M the number of trees generated by the algorithm. It is important for the variance reduction of the final model. Once M trees are generated, their predictions are aggregated to compute the final prediction, by using the so called "majority vote" in classification problems and by calculating the arithmetic average in regression problems. This approach is powerful because it should be able to reduce variance more than any other method. In addition, the use of all the training samples is motivated in order to minimize the bias.

In the limit of $M \to \infty$, one can show that for any N-dimensional space and $N_{\min} \geq 2$, it produces a continuous piecewise multi-linear approximation of the sample. It means that, when $M \to \infty$, the model is much more accurate and smooth than other tree based algorithms. To give an idea of it, let's consider a sample of data

$$X = \{(x^i, y^i) | i = 1, \dots, n\},$$

where $x^i = (x^i_1, \dots x^i_N)$ is a data point with N features and y^i is the output label value. Then, denote with

$$(x_j^{(1)}, \dots, x_j^{(n)}) \tag{4.24}$$

the sorted sample values (increasingly) restricted to the $j-$th column related to the $j-$th feature. Then, for notation, we indicate with $x_j^{(0)} = -\infty$ and $x_j^{(n+1)} = +\infty$ for all $j = 1, \dots, N$ and we denote, $\forall (i_1, \dots, i_N) \in \{0, \dots, n\}^N$, by $I_{(i_1,\dots,i_N)}(x)$ the characteristic function of the rectangle

$$[x_1^{(i_1)}, x_1^{(i_1+1)}[\times \cdots \times [x_N^{(i_N)}, x_N^{(i_N+1)}[. \tag{4.25}$$

By using this formalism, it is possible to prove that, for $M \to \infty$ that means for

an infinite set of trees, the model achieve the approximation in the form

$$\hat{y}(x) = \sum_{i_1=0}^{n} \cdots \sum_{i_N=0}^{n} I_{(i_1,\ldots,i_N)}(x) \left(\sum_{\mathcal{X} \subset \{x_1,\ldots,x_N\}} \lambda^{\mathcal{X}}_{(i_1,\ldots,i_N)} \prod_{x_j \in \mathcal{X}} x_j \right), \tag{4.26}$$

where $\lambda^{\mathcal{X}}_{(i_1,\ldots,i_N)}$ are real parameters valued through the training set which depend on $n_{\min}$ and K. Observe that, when $n_{\min} = 2$ the equation 4.26 is such that $\hat{y}(x^i) = y(x^i)$ for all (x^i, y^i) belonging to the initial set. In addition, if the space is monodimensional, i.e. $n = 1$, then it's easy to note the linearity of the model, since equation 4.26 becomes

$$\hat{y}(x) = \sum_{i_1=0}^{n} I_{(i_1)}(x) \left(\sum_{\mathcal{X} \subset \{x_1\}} \lambda^{\mathcal{X}}_{(i_1)} \prod_{x_j \in \mathcal{X}} x_j \right) = \sum_{i_1=0}^{n} I_{(i_1)}(x_1) \left(\lambda_i^{\emptyset} + \lambda_i^{\{x_1\}} x_i \right). \tag{4.27}$$

The values of λ parameters can be computed directly from the n equations $\hat{y}(x^i) = y(x^i)$ and the constraints. In conclusion, extremely randomized tree ensembles provide an interpolation of any output variable which, for finite M is piecewise constant, and for $M \to \infty$ becomes piecewise multi-linear and continuous.

4.4.2 Importance Analysis

Extremely Randomized Trees consist of many decision trees, each of them built over a random extraction of the features, using all the available observations. Not every tree sees all the features and this guarantees that the trees are uncorrelated and so further away from overfitting. Each tree is also a sequence of 0/1 results based on a single or combination of features. At each node, the Extremely Randomized Trees divides the dataset into 2 buckets, each of them hosting observations that are more similar among themselves and different from the ones in the other bucket. Therefore, the importance of each feature is derived from how "pure" each of the buckets is. For the measure of this pureness we decide to use the entropy score, as it is defined in a previous section.

So, time label gap by time label gap we test our Extremely Randomized Trees in order to estimate the importance of each SPN (feature) and make a ranking. An

example of feature importance ranking by ERT is shown in Figure 4.5, where the time label considered is $30.0h$. Since we want only those features which explain as

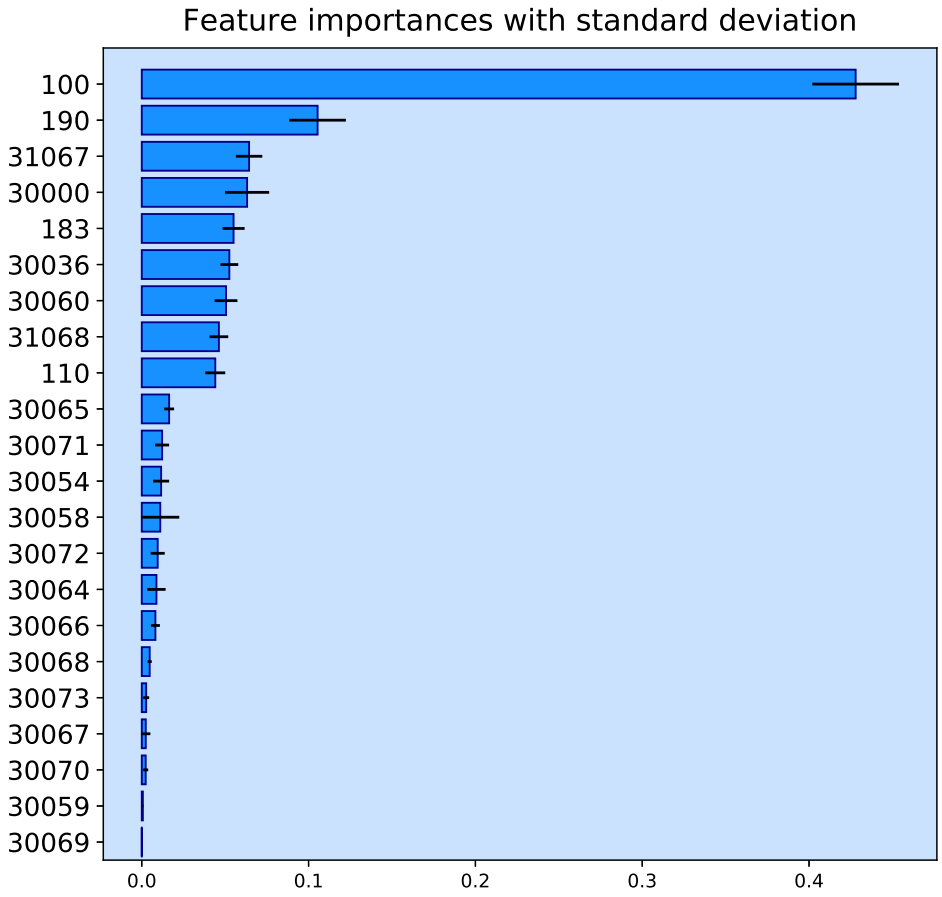


Figure 4.5: Bar graph showing for each SPN its mean importance computed by ERT, together with its standard deviation. The label used is $30.0h$.

more variance as possible, we set a threshold. So, if the importance of one SPN is greater than or equal to the threshold, then it remains and it's not filtered. The threshold is set to 10.0% and the results are shown in the table 4.2.

From the table we see that with this approach two out of all the SPNs are relatively important for the model: SPN 100 and 190. But even if the first one is

	label_time	feature	importance_ETC
0	3.0h	100	42.51
1	3.0h	190	10.21
2	6.0h	100	42.11
3	9.0h	100	42.61
4	9.0h	190	10.11
5	12.0h	100	43.07
6	12.0h	190	10.47
7	15.0h	100	42.84
8	15.0h	190	10.47
9	18.0h	100	42.73
10	21.0h	100	42.43
11	24.0h	100	42.37
12	24.0h	190	10.63
13	27.0h	100	42.43
14	27.0h	190	10.44
15	30.0h	100	42.39
16	30.0h	190	10.75
17	33.0h	100	42.10
18	33.0h	190	11.02
19	36.0h	100	42.35
20	36.0h	190	10.58
21	39.0h	100	42.09
22	39.0h	190	11.09
23	42.0h	100	41.40
24	42.0h	190	10.84
25	45.0h	100	40.84
26	45.0h	190	10.69
27	48.0h	100	40.47
28	48.0h	190	10.62
29	72.0h	100	37.05
30	72.0h	190	11.07
31	96.0h	100	34.06
32	96.0h	190	10.84
33	120.0h	100	30.92
34	120.0h	190	11.46
35	144.0h	100	28.74
36	144.0h	190	11.22

Table 4.2: Table with importance scores for time label gaps and features, by ERT.

dominating the SPN 190, it is totally unexpected the occurrence of the engine speed into the most significant parameters to predict the anomaly engine oil pressure.

4.5 Creating a new feature: Moving Average Engine Oil Pressure

A great limitation of our problem is that we use time series with a lot of noise. In general, it might be very hard to guess a method that could make time series as perfect and clean as we can create a model with no noise. Usually this inability happens because we can't use the same method for all the different kind of time series. In many cases, the information we provided doesn't help us to get a sure decision. As we said in the previous chapter, a method that we can use is the moving average. Since the important SPN is 100, its behaviour is suitable to apply this method.
Feature creation is that part of machine learning that needs human intervention in creatively finding the best approach to mix or derive features in order to create a new one more accurate and useful.

So, before going on with the last uncommon method for feature selection, let's observe that most likely the only one SPN weakly correlated with the DM1 occurrences is the SPN 100. In fact we will do an intersection of all the results, such as the parameters which pass all the three tests are significant for real. Actually, it misses another one important test and for this reason an important decision has to be taken. As the time series related to the engine oil pressure is very fluctuating, we feel that it would be important to correct just a little this characteristic adding a new column with the moving average of SPN 100. It is necessary to separate as well as we can the two different behaviours of that SPN. Through this new factor the sequenciality could be more relevant and we might obtain better results. So, the new feature is called "100_ma" which stands for moving average of engine oil pressure and it is created by using the formula 3.12 with $q = 20$. The choice of q is related to the data points: looking into the values, we can see that, when it starts changing because of a unit problem, about 20 timestamps are different from the

general behaviour of the parameter. So, as 20 is large enough to clear partially the noise, we decide to fix this value and leave eventually its optimal choice at the end of the work. The result can be seen in Figure 4.6.

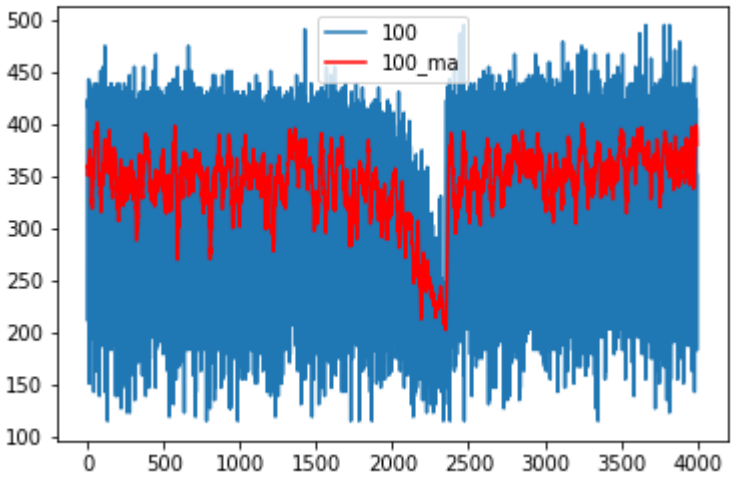


Figure 4.6: Graph showing the reduction of noise obtained using the moving average (100_ma).

Now we are able to carry on and finish this very significant part about feature selection, testing all the SPNs present in the dataset and the new moving average SPN.

4.6 Dynamic Time Warping

When a comparison between time series is required, measurement functions provide meaningful scores to characterize similarity between two or more sequences. Usually, time series are warped in time, i.e, although they may exhibit shape similarity, they appear dephased in time. A very simple distance measure, such as Euclidean distance will suffice in the case that two sequences have approximately the same overall component shapes. Unfortunately, as mostly of the times, our time series have different shapes from each other. In order to find the similarity between

such sequences we must "warp" the time axis of one (or both) sequences to achieve a better alignment. The most common algorithm to overcome this challenge is the Dynamic Time Warping, which is a technique for achieving the warping efficiently. It aligns each sequence prior establishing distance measurements. For this section we refer to articles [Fur08; KP01; Fol+18; SC07; SZZ15; Rak+12].

4.6.1 Dynamic Time Warping Algorithm

Suppose now that we have two time series X and Y, of length n and m respectively, where:

$$X = x_1, x_2, \ldots, x_n \coloneqq X(t_1), X(t_2), \ldots, X(t_n);$$

$$Y = y_1, y_2, \ldots, y_m \coloneqq Y(s_1), Y(s_2), \ldots, Y(s_m).$$

To align two sequences using DTW we create an $n - by - m$ matrix where the general element (r, s) of the matrix D contains the distance $d(x_r, y_s)$ between the two p-dimensional points x_r and y_s. In this thesis, as most of the times, the Euclidean distance is used, so that

$$d_{r,s} = d(x_r, y_s) = \|x_r - y_s\|_2, \tag{4.28}$$

and

$$D = \{d_{r,s} : r \in \{1, \ldots, n\}, s \in \{1, \ldots, m\}\}.$$

Each matrix element (r, s) corresponds to the alignment between the points x_r and y_s. A warping path w, is a contiguous (in the sense stated below) set of matrix elements that defines a mapping between the series X and Y. By defining the k-th element of w as $w_k = (r, s)_k$, we obtain that:

$$w = (w_1, w_2, \ldots, w_k, \ldots, w_K)$$

with

$$\max(n, m) \leq K < n + m - 1.$$

An example of how DTW works is shown in the figure 4.7[1].

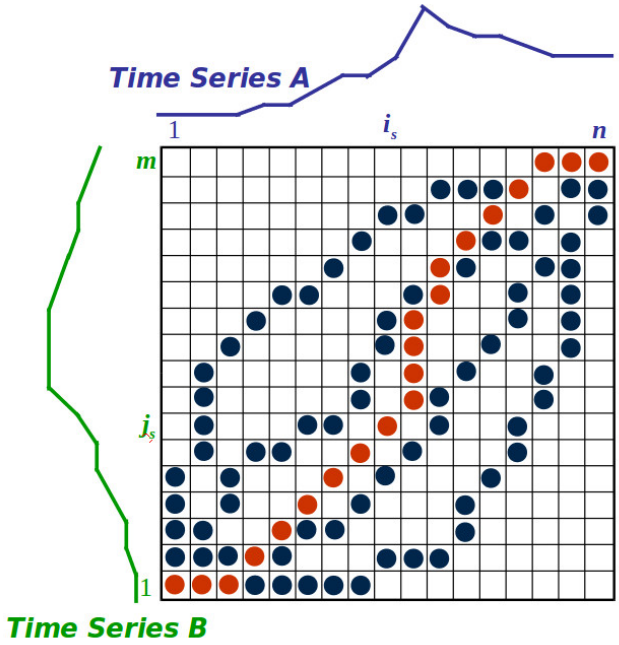


Figure 4.7: Two different time series are represented (green and blue). It is shown what is a path (black) and what will be defined as the optimal path (red).

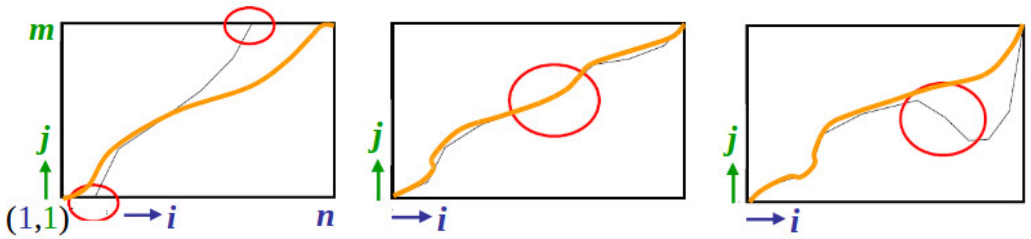
The warping path is typically subject to several constraints:

- *Boundary condition*: $w_1 = (1, 1)$ and $w_K = (n, m)$, this requires the warping path to start and finish in diagonally opposite corner cells of the matrix (Figure 4.8a)[2].

- *Continuity condition*: Given $w_k = (a, b)$ then $w_{k+1} = (a', b')$ where $a - a' \leq 1$ and $b - b' \leq 1$. This restricts the allowable steps in the warping path to adjacent cells, including diagonally adjacent cells (Figure 4.8b)[3].

[1] http://ros-developer.com/2017/11/17/ros-packages-for-dynamic-time-warping/.
[2] http://ros-developer.com/2017/11/17/ros-packages-for-dynamic-time-warping/.
[3] http://ros-developer.com/2017/11/17/ros-packages-for-dynamic-time-warping/.

- *Monotonicity condition*: Given $w_k = (a, b)$ then $w_{k+1} = (a', b')$ where $a - a' \geq 0$ and $b - b' \geq 0$. This forces the points in w to be monotonically spaced in time (Figure 4.8c)[4].



(a) (b) (c)

Figure 4.8: At the left, figure (a) illustrates that the alignment path must start at the bottom left and end at the top right. In the centre , figure (b) shows that the alignment path must not jump in time index (continuity). At the right, figure (c) exposes that the alignment path shall not go back in time index (monotonicity).

Let's sum up what we have said above formalizing what is a *warping path* in a definition:

Definition 4.6. A *warping path* is a sequence

$$w = (w_1, w_2, \ldots, w_k, \ldots, w_K)$$

where for k in $\{1, \ldots, K\}$,

$$w_k = (r, s)_k = (r_k, s_k)$$

with $r_k \in \{1, \ldots, n\}$ and $s_k \in \{1, \ldots, m\}$, satisfying the *boundary*, *continuity* and *monotonicity* conditions.

For simplicity, we write $w = (r, s)$ as a path in accordance with definition 4.6.

There are exponentially many warping paths that satisfy the above conditions, however we are interested only in the path which minimizes the total distance. Luckily, this path can be found very efficiently. In fact, the optimal warping path

[4] http://ros-developer.com/2017/11/17/ros-packages-for-dynamic-time-warping/.

is the path that has the minimum total cost among all possible warping paths for a certain defined cost function. One could test every kind of warping path and determine the minimum cost candidate, but such method will lead to a exponential computational complexity in the lengths of n and m. Using *dynamic programming* we can compute an accumulated cost matrix C in order to find the path that minimizes the warping cost in an $O(nm)$ complexity. The time cost of building this matrix equals the cost of the algorithm 3, where X and Y are the input time series and D is the local cost matrix representing all the pairwise distances between X and Y.

input : X and Y, two time series of length n and m,
D, the local distance/cost matrix;
output: C, the accumulated cost matrix.

$n \leftarrow |X|$;
$m \leftarrow |Y|$;
$C \leftarrow new[n \times m]$;
$C(0,0) \leftarrow 0$;
for $i = 1;\ i \leq n;\ i++$ **do**
  $C(i,1) \leftarrow C(i-1,1) + D(i,1)$;
end
for $j = 1;\ j \leq m;\ j++$ **do**
  $C(1,j) \leftarrow C(1,j-1) + D(1,j)$;
end
for $i = 1;\ i \leq n;\ i++$ **do**
  for $j = 1;\ j \leq m;\ j++$ **do**
    $C(i,j) \leftarrow D(i,j) + \min\{C(i-1,j); C(i,j-1); C(i-1,j-1)\}$;
  end
end
return C

Algorithm 3: Accumulated Cost Matrix

Coming back to a more formal part, we have to define the cost function and the Dynamic Time Warping Distance with much more accuracy.

Definition 4.7. Let X and Y be two time series with local distance matrix D.

Then, for a fixed path $w = (r, s)$, we define the *cost* c as:

$$c(w) = \sum_{k=1}^{K} D_{r_k, s_k}.$$

Definition 4.8. The Dynamic Time Warping Distance (DTWD) is the sum of the pointwise distances along the optimal path $\hat{w}$, for the cost function formulated in definition 4.7:

$$\hat{w} = \arg\min_{w} c(w), \quad DTWD(X, Y) = c(\hat{w}).$$

Basically, starting with local distances matrix D, then the minimal distance matrix between sequences is computed using a dynamic programming algorithm and the following optimization criterion:

$$\hat{c}_{r,s} = d_{r,s} + \min(\hat{c}_{r-1,s-1}, \hat{c}_{r-1,s}, \hat{c}_{r,s-1}), \tag{4.29}$$

where $\hat{c}_{r,s}$ is the minimal distance between the sub-sequences $x_1, x_2, \ldots, x_r$ and $y_1, y_2, \ldots, y_s$. It means that each element of the accumulated cost matrix is defined as the local cost measure in the current cell plus the minimum of the local distance/cost measures in the adjacent cells.

Once the accumulated cost matrix has been built, the optimal warping path could be found by the simple backtracking from the end point with coordinates (n, m) to the first one with coordinates $(1, 1)$ following the greedy strategy as described by the Algorithm 4.

One time series X can be aligned to another Y by using the optimal path defined in Definition 4.8. The basic concept of DTW alignment is illustrated in Figure 4.9.

4.6.2 DTW Distance for sequentiality Feature Selection

In this section we are going to explain how can we use DTW distance for our problem. First of all, it's necessary to keep in mind that the task we want to

input : C, the accumulated cost matrix;
output: *path*, the optimal warping path.

```
path[ ] ← new array;
i = rows(C);
j = columns(C);
while (i > 1) and (j > 1) do
    if i == 1 then
        j = j − 1;
    end
    else if j == 1 then
        i = i − 1;
    end
    else
        if C(i−1, j) == min{C(i−1, j); C(i, j−1); C(i−1, j−1)} then
            i = i − 1;
        end
        else if C(i, j−1) == min{C(i−1, j); C(i, j−1); C(i−1, j−1)}
         then
            j = j − 1;
        end
        else
            i = i − 1;
            j = j − 1;
        end
    end
    path.add((i, j));
end
return path
```

Algorithm 4: Optimal Warping Path

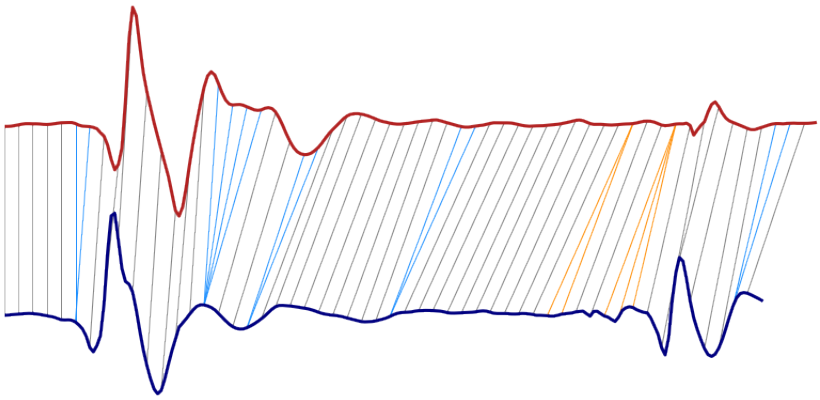


Figure 4.9: The alignment of two different time series (red and blue) using DTW. The image is taken from the site http://josejg.com/.

achieve concerns the feature selection. In a certain way it could be classified as a first method to predict a DM1 observation, even if the main role of this part is to understand which CAN parameters are strictly connected with DM1 occurrences. This filter based method finds the most important parameters by analysing them individually. After having used three approaches based on CAN values, we continue checking if there exist a real dependence between CAN parameters and the DM1 selected, from the point of view of sequentiality of values. Basically, the main idea is to use the time series distance as a measure of dependence between features and the predictive label.

Remind that the proposed method splits the data of one parameter into two groups according to the time distance from the next diagnostic message. The first group includes all data labelled as "Normal" while the second one includes all data labelled as "Faulty". We them split the first group into many sub-groups: once we have computed the length of the data labelled as "Faulty", we divide "Normal" data into the maximum number of groups having the same length as the faulty group, as reported in Figure 4.10. In order not to create an enormous number of groups, we set the maximum value at 200. If the maximum is reached, for example because "Normal" data more than 300 times the "Faulty" ones, it makes sense to

consider just the last 200 sub-groups. It means that we take "Normal" data from the end of the list until all the groups are composed. This decision comes from the idea that farther data are more affected by noise then the others. In addition, it is necessary to select just a portion of groups (200 in this case) to make the model not too slow: since we are going to compute many distances, this process might request a lot of time to be processed.

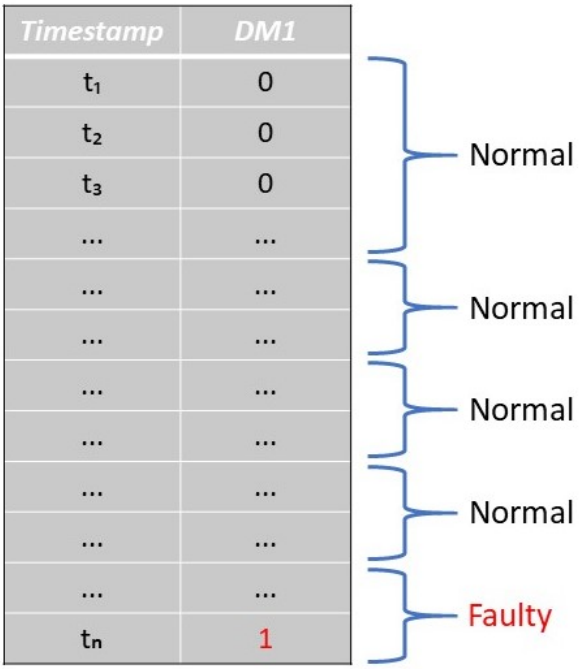


Figure 4.10: Splitting CAN messages into Normal and Faulty groups.

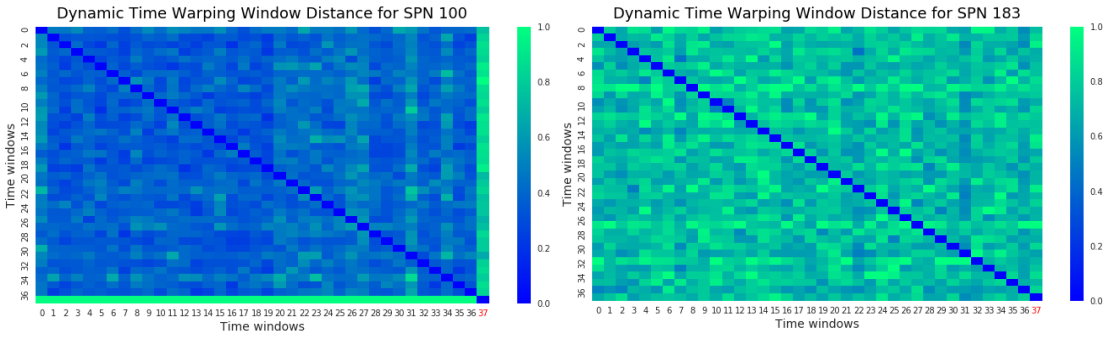
In this way many time series are obtained. The zero-task is to understand if the future presence of the DM1 is strictly correlated, in a wide sense that we are going to explain, with a change in the time of the CAN messages. So, if there exists a real dependence between them, hopefully, we will find that the maximum distance between each time series and all the others (one at a time) will be reached in correspondence of the "Faulty" one or, at least, in the nearby. In general, we are going to estimate the variation of CAN parameters during the time in the sense of sequentiality. In fact, if the CAN messages from a certain period of time corresponds to the time series from which all the other groups have the maximum

possible distance, then it means in that period those CAN messages have had a mutation, and it could be connected to the appearance of a DM1.

Finally, we are going to repeat this computation many times, using time windows for labelling data set to different values. The way we choose those time windows is very basic and a little bit naive: since we don't know what's the impact of the DM1 on the vehicle (and so when CANs start changing), we use a linear generator of windows. It starts labelling as "Faulty" all the CAN from three hours before until the time the DM1 appears and as "Normal" all the CANs before the first "Faulty", according to the previous consideration (until 200 groups are formed). The next "Faulty" window is wide 6 hours, then 9 hours and so on until a maximum of 48 hours before. The reason and the goal of using many window labels concerns the kind of correlation between SPNs and the DM1: it might be possible that one SPN is very connected with the error message and it starts changing many hours before, whereas an other one has a little mutation very close it (and for this little change the DM1 appears).

Let's summarize: we consider the first timestamp when the DM1 appears; then we divide CAN messages into two different labelled groups, the "Normal" and the "Faulty" one. After that, we can create as many time series inside the "Normal" group as we can (until 200 groups) using the time window length equal to the number of "Faulty" CANs. For example, if there are 50.000 messages in the "Normal" group and 1000 inside the other one, we should create exactly 50 different time series inside the first one, partitioning normal data in order to hold in each group 1000 timestamps.

Finally, it's possible to measure the change in time of those CANs in term of sequentiality: computing the DTW distance between each pair of time series, we are going to obtain, as a result, one by one which time series is the furthest from those CAN messages. In fact, as we have seen in the theoretical part, while computing the DTW distance the first time series is matched into the other one by a process of "all VS all". It means that the algorithm matches all the subsequences of the first time series into all the parts of the other one, looking for similarities.



(a) (b)

Figure 4.11: Heatmap showing for each window time series its DTW distance from each one of the others. Lighter colors mean higher distance values. At the left, figure (a) illustrates an example of DTW time window for SPN 100. At the right, figure (b) illustrates DTW for SPN 183. In this case the number of windows is 38.

DTW-FS results

Let $X_1, \ldots, X_n$ be the time series created as explained in the previous chapter, where $n \leq 200$. Furthermore, let's keep in mind that the faulty time series is the last one (X_n). In order to visualize better the main support idea of this approach, in figure 4.11 are shown two examples of two opposite situations. First of all, let's consider a unit and a DM1 occurrence. Then, proceeding as above, we can print a heat map showing for each time window the distance value from all the others. Let's comment the results in the picture 4.11. The first image represents the heat map using mono-dimensional time series with SPN 100, which is directly connected with the DM1 and so it will reach high scores. The other involves one SPN which is probably not dependent of the response variable. The two heat maps are completely different: the first shows high values only in the last column (or row), meaning that the related window time series is the farthest from the others. On the other hand, the second one has uniform chaotic values, meaning that there is no farthest window inside.

For each sequence X_i we are going to compute the distance with all the others, so that the result will be represented in a distance matrix W, such that

$$W_{i,j} = DTWD(X_i, X_j) \quad i, j \in \{1, \ldots, n\},\ n \leq 200.$$

Then, for each row of W we will save just the index (and the timestamp start/end) of the time series that corresponds to the maximum distance. It means that for each sequence X_i, create the set

$$K_i = \{j \in \{1, \dots, n\} : W_{j,i} = \max\{W_{j,k} : k \in \{1, \dots, n\}\} \tag{4.30}$$

Then, fixed an index i, the set K_i in 4.30 indicates from how many sequences the time series X_i reach the maximum distance, in the sense that it has the deepest sequentiality change. A particular look has to be done at the index n, which we could think it might have the largest set K_n. In addition, a basic statistic we can use is the rate

$$r_i = \frac{|K_i|}{n-1} \quad i \in \{1, \dots, n\} \tag{4.31}$$

where $|\cdot|$ stands for the cardinality of the set. It indicates the rate of the amount of the indexes in K_i with respect to the total number of time series. Then, for each vehicle, time label and DM1 observation we are going to save only the time sequence with the highest rate. Furthermore, since we would want to measure if the mutation of data can cause the DM1, we are particularly interested in the faulty sequence. Then we are going to count the number of them still present and call this number with "n Faulty CAN". As we are testing this method over different time window lengths and we only care about it, we are going to group the results over units and number of DM1 observations, adding values time window label by time window label. First of all we suppose all the time series to be p-dimensional, where p indicates the number of available CAN parameters. In fact, at the beginning we want to find if there exist a multidimensional correlation from the point of view of sequentiality. For this case, the results are summarised in the table 4.3. The "Count" column represents the total number of test we do, that is the number of times the DM1 appears non consecutively. The last column "Rate Faulty/Normal" is the fraction between the number of "Faulty CAN" and the "Count". We define a faulty time series as *significant* if its rate Faulty/Normal is ≥ 0.75. In table 4.3 there are no significant faulty time series, that means the $p-$dimensional CAN parameter doesn't change a lot close to the DM1.

	Time label	n Faulty CAN	Count	Rate Faulty/Normal
0	3.0h	0.0	11	0.000000
1	6.0h	0.0	9	0.000000
2	9.0h	0.0	9	0.000000
3	12.0h	0.0	9	0.000000
4	15.0h	0.0	9	0.000000
5	18.0h	0.0	9	0.000000
6	21.0h	0.0	9	0.000000
7	24.0h	1.0	8	0.125000
8	27.0h	1.0	8	0.125000
9	30.0h	1.0	7	0.142857
10	33.0h	0.0	6	0.000000
11	36.0h	0.0	6	0.000000
12	39.0h	1.0	6	0.166667
13	42.0h	1.0	6	0.166667
14	45.0h	2.0	6	0.333333
15	48.0h	2.0	6	0.333333

Table 4.3: Table to test correlation between SPN and DM1 (multidimensional).

Subsequently, known that all the SPN together don't correlate with the error, we are going to filter only those SPNs which really have an impact on analysis. So, dimension by dimension let's repeat this sort of test and let's report the final table, obtained after filtering the significant time series. In 4.4 is shown the list of SPNs and time labels passing the test. We can see that only two SPNs are strongly correlated with the DM1. For this two features, we use the same optimal time label which is the first in common of both: $30.0h$. So, for the rest of the analysis we are going to make predictions using just these two SPNs, which correspond to the engine oil pressure and its moving average.

4.7 Final subset of SPNs for engine oil pressure analysis

Currently we have all the results from each method we have used. So, the last part has the role to group together the informations and extract the final subset of features. However, it is not a simple job because of the very weak results. In fact, choosing the wrong thresholds would make different features be selected. Generally

	spn	Time label	n Faulty CAN	Count	Rate Faulty/Normal
0	100	27.0h	8.0	8	1.000000
1	100	30.0h	7.0	7	1.000000
2	100	33.0h	6.0	6	1.000000
3	100	39.0h	6.0	6	1.000000
4	100	42.0h	6.0	6	1.000000
5	100	45.0h	6.0	6	1.000000
7	100_MA	30.0h	6.0	7	0.857143
8	100_MA	33.0h	5.0	6	0.833333
9	100_MA	39.0h	5.0	6	0.833333
10	100_MA	42.0h	6.0	6	1.000000
11	100_MA	45.0h	6.0	6	1.000000

Table 4.4: Table to test correlation between SPN and DM1 (monodimensional).

talking about feature selection, it is extremely relevant either to minimize the curse of dimensionality or to help deal with overfitting, just to explicitly claim two different ways to explain the issue of excessively complex modeling.
However, there is not way to find the perfect and exact threshold to select a feature. So, as we want to point out the uselessness of most of SPNs, the thresholds will not be relaxed to force the occurrence of more SPNs.

By intersecting all the previous results we obtain a dataframe of which a sample of 15 rows is reported in table 4.5.

Then, by applying the thresholds, we reach the goal to have the set with just the relevant features. For the sake of honesty, we report once again the threshold we put:

- Mutual information score (MI (%)) has to be greater then 0.5;
- Importance of Extremely Randomized Classifier (ETC (%)) has to be greater than 10.0;
- Recursive Feature Elimination Cross Validation ranking (RFECV) has to be exactly 1.0;
- Dynamic Time Warping Faulty Normal rate (DTW (%)) has to be greater than 85.0.

label_time	feature	MI (%)	ETC (%)	RFECV	DTW (%)
3.0h	30000	0.00	6.35	5.0	0.0
3.0h	30058	0.61	0.89	3.0	0.0
9.0h	100	1.87	42.61	4.0	77.8
9.0h	110	0.27	4.37	4.0	11.1
15.0h	30058	0.56	0.98	1.0	0.0
18.0h	183	0.11	5.47	1.0	0.0
21.0h	30064	0.51	0.91	1.0	0.0
30.0h	30000	0.00	6.05	1.0	28.6
33.0h	30072	0.07	0.96	1.0	0.0
33.0h	30073	0.47	0.36	1.0	0.0
36.0h	110	0.28	4.49	1.0	33.3
39.0h	30065	1.00	2.11	1.0	0.0
42.0h	30064	0.59	0.99	1.0	0.0
45.0h	100	3.25	40.84	1.0	100.0
45.0h	110	0.26	4.35	1.0	16.7

Table 4.5: Table showing the final dataframe with all the scores for feature selection. Note that RFECV column has just a ranking, whereas the others have the exact score.

So, the final result is shown in table 4.6. It is clear that there is only one SPN significant for the DM1 and it is the 100. Obviously, its moving average is out of this final result, since it has been considered only for the DTW score.

From this point on, we are going to use only two features to make the model as accurate as we can: SPN 100 and the derived moving average feature 100_ma. Furthermore, because of all the time labels between 21 hours and 45 hours are relevant, we decide to use as default one the 30.0h which is a mid time. This choice is motivated by the results of the following methods, which are slightly higher than the others.

label_time	feature	MI (%)	ETC (%)	RFECV	DTW (%)
21.0h	100	2.23	42.43	1.0	88.9
24.0h	100	2.49	42.37	1.0	87.5
27.0h	100	2.78	42.43	1.0	100.0
30.0h	100	2.94	42.39	1.0	100.0
33.0h	100	3.08	42.10	1.0	100.0
36.0h	100	3.08	42.35	1.0	100.0
39.0h	100	3.13	42.09	1.0	100.0
42.0h	100	3.19	41.40	1.0	100.0
45.0h	100	3.25	40.84	1.0	100.0

Table 4.6: Table showing the final dataframe containing only those relevant features and the respective time label.

Chapter 5

DM1 prediction: a probabilistic approach

5.1 Discrete Time Markov Chains

In order to figure out the reasons why Markov models could be useful for our purpose, let's start giving some general definitions, as in [DD99]. Let $\{X_t\}_{t\in T}$ be a stochastic process.

Definition 5.1. A *discrete time stochastic process with discrete state space* S is a collection of random variables $\{X_t\}_{t=0}^{\infty}$ with the property that the range of X_t is the discrete space S, for every $t \in \mathbb{N}$. We say that S is a discrete space if $S = \{S_1, S_2, \ldots, S_N\}$ or $S = \mathbb{N}$.

If $S = \{1, 2, 3\}$, then one example of transition graph of the process is shown in the figure 5.1. In addition, a possible time evolution of the process, given that $X_0 = 1$ is shown in the picture 5.2.

Definition 5.2. Let $\{X_t\}_{t=0}^{\infty}$ be a discrete time stochastic process with a discrete state space S. We say that $\{X_t\}_{t=0}^{\infty}$ is a *discrete time Markov chain* if it has the Markov property, that means for any $j, i, i_{t-1}, \ldots, i_0 \in S$

$$\mathbb{P}\left(X_{t+1} = j | X_t = i, X_{t-1} = i_{t-1}, \ldots, X_0 = i_0\right) = \mathbb{P}\left(X_{t+1} = j | X_t = i\right). \qquad (5.1)$$

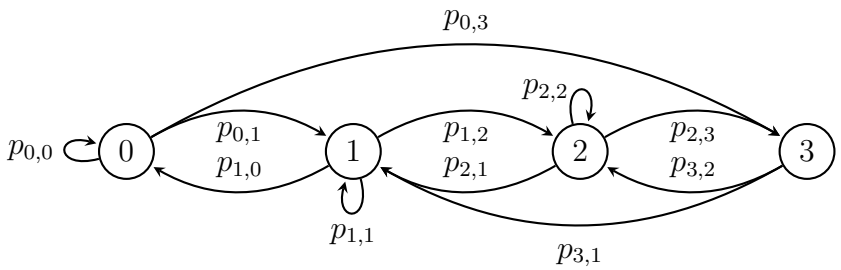


Figure 5.1: Example of Discrete Time Markov Chains with four states.

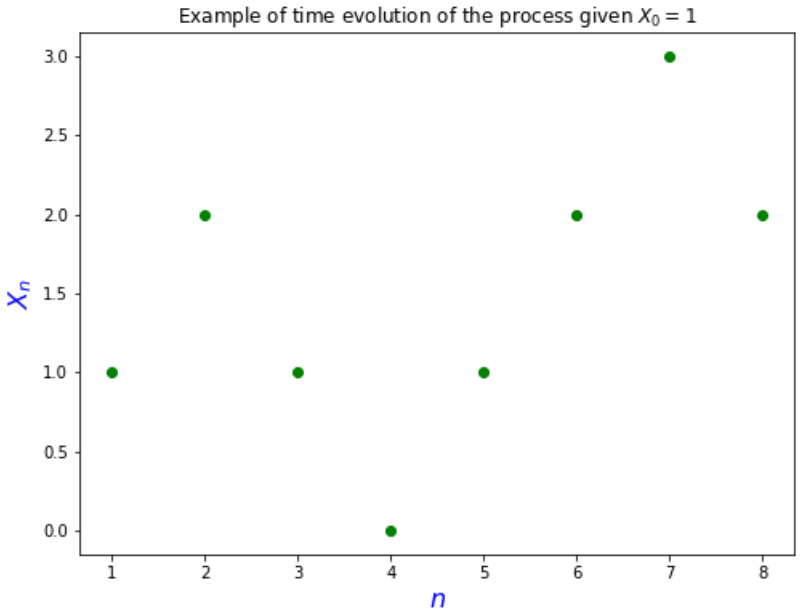


Figure 5.2: Example of Discrete Time Markov Chain time evolution.

From this point on we will indicate the one step transition probabilities with

$$p_{i,j} \coloneqq \mathbb{P}\left(X_{t+1} = j | X_t = i\right). \tag{5.2}$$

The identity 5.1 tells us that the probabilities associated with future states only depends on the current state, and not on all the past states of the process. Basically, it means that the future depends on the past only through the present. Note that, in the case we are going to consider, the transition probabilities are not time dependent. This particularly case is called homogeneous. When the state space is finite we can write the one step probabilities $p_{i,j}$, defined in 5.2, in a finite matrix P, with entries $p_{i,j}$. This matrix is called transition matrix (or transition probability matrix). It satisfies two relevant properties:

- $0 \leq P(i,j) \leq 1$, for all $i, j \in S$;
- $\sum_{j \in S} P(i,j) = \sum_{j \in S} p_{i,j} = 1$, for all $i \in S$.

The above stochastic process $\{X_t\}_{t=0}^{\infty}$ could be called an observable Markov model since the output of the process is the set of states at each instant of time, where each state corresponds to an observable event.

For each $i \in S$, let π_i be the probability of the system or object to be in state i at time $t = 0$, where it is assumed that

- $\pi_i \in [0,1]$;
- $\sum_{i \in S} \pi_i = 1$.

The vector $\pi = (\pi_1, \ldots, \pi_M)^\top$ of the probabilities $\pi_1, \ldots, \pi_M$ defines the initial distribution of the Markov chain.

5.2 Intuition behind Hidden Markov Models

Hidden Markov models are a general statistical modeling technique for "linear" problems like sequences or time series and have been widely used in a lot of different scenarios. Within the HMM formalism, it is possible to apply formal, fully

probabilistic methods to profiles and gapped sequence alignments. The key idea is that an HMM is a finite model that describes a probability distribution over an infinite number of possible sequences [Sal05; KD; Sta04; Rab89; Ye+16; YXC94].

Considering Markov models as if they corresponded to an observable event in each state is too restrictive to be applied to many problems. So, the common Model has to be extended in order to include cases where the observation is a probabilistic function of the state. It means that the resulting model is a doubly embedded stochastic process with an underlying stochastic process that is not observable (for this reason it is called hidden), but it can only be observed through another set of stochastic processes that produce the sequence of observations. To have a better idea of what a HMM is and how it can be applied to our scenario, we now formally define the elements of an Hidden Markov Model and explain how the model generates observation sequences. It is composted by:

- N, the *number of states* in the model. Even if these states are hidden, in many practical applications there is a meaning attached to each state or, more in general, to sets of states of the model. Usually the model is *ergodic*, in the sense that all the states are interconnected in such a way that any of those can be reached from any other state. We denote the individual states as $S = \{S_1, S_2, \ldots, S_N\}$ and the state at time t as X_t.

- M, the *number of distinct observation symbols* per state, which correspond to the physical output of the modeled system. We denote the individual symbols as $V = \{v_1, v_2, \ldots, v_M\}$. The set of observation symbols is sometimes called the alphabet of the HMM.

- The *state transition probability distribution* $P = \{p_{i,j}\}$, where $p_{i,j}$ is defined in 5.2. For the special case where we assume that any state can reach any other state in one single step, we have that $p_{i,j} > 0$ for all i, j. Otherwise, in all the other cases, we would have $p_{i,j} = 0$ for one or more couple of states.

- The observation symbol probability distribution in each state $j \in \{1, \ldots, N\}$.

This is called *emission matrix*, $B = \{b_j(k)\}$, where

$$b_j(k) = \mathbb{P}\left[v_k \text{ at time } t | X_t = S_j\right] \qquad 1 \leq j \leq N,\ 1 \leq k \leq M. \tag{5.3}$$

- The *initial state distribution* $\pi = \{\pi_i\}$, where

$$\pi_i = \mathbb{P}\left[X_1 = S_i\right], \qquad 1 \leq i \leq N. \tag{5.4}$$

So, given N, M, P, B, π we can use the HMM to generate sequences like

$$O = o_1 o_2 \ldots o_T$$

where $o_t \in V$ and T is the number of observations inside the sequence. The procedure to create that sequence is, basically,

1. Choose the initial state $X_1 = S_i$, according to π and set $t = 1$.
2. Extract $o_t = v_k$ using $b_i(k)$, the symbol probability distribution of the state S_i.
3. Change state to a new one $X_{t+1} = S_j$ through $p_{i,j}$, the transition probability distribution.
4. Increment t and continue if $t < T$, terminate the procedure otherwise.

From now, we are going to write the model with a compact notation $\lambda = (P, B, \pi)$ to indicate the complete parameter set, without explicitly stating N and M.

5.3 Three Basic Problems for HMMs

Given a Hidden Markov Model λ, we can consider three problems connected to its real application. Let's consider a sequence of observations $O = o_1 o_2 \ldots o_T$. The three questions are the following:

1. How can we do to compute efficiently $\mathbb{P}(O|\lambda)$, that is the probability of the sequence given the model?

2. How can we choose the corresponding state sequence $X = X_1 X_2 \dots X_T$ which is optimal in some meaningful sense?

3. How do we adjust the model parameters to maximize $\mathbb{P}(O|\lambda)$?

The first question concerns the evaluation problem: given a model and a sequence of observations, how do we compute the probability that the exact sequence was produced by our model? The second problem regards the attempt to find the correct state sequence, looking into the hidden states. This is sometimes called a *decoding problem*. The last one is about training problem and the related method to optimize the model parameters to describe as well as possible how a given observation sequence comes about. It is called either *learning problem* or *optimization problem*.

At the beginning, we would want to compute the probability of an observation sequence O, as we said above, given the model λ. A possible but not so efficient way of doing this is through enumerating all the feasible state sequence of length T. Then, let's fix one of them $X = X_1 X_2 \dots X_T$, where X_1 is the first state, X_2 the second one, and so on. The probability of the sequence (of observations) O for the fixed state sequence is

$$\mathbb{P}\left(O|X,\lambda\right) = \prod_{t=1}^{T} \mathbb{P}\left(O_t|X_t,\lambda\right) \tag{5.5}$$

where we assume that each observation is independent from the others. So, in this case

$$\mathbb{P}(O|X,\lambda) = b_{X_1}(o_1) \cdot b_{X_2}(o_2) \cdot \dots \cdot b_{X_T}(o_T). \tag{5.6}$$

Since the probability of the state sequence can be written as

$$\mathbb{P}(X|\lambda) = \pi_{X_1} \cdot p_{X_1,X_2} \cdot p_{X_2,X_3} \cdot \dots \cdot p_{X_{T-1},X_T}, \tag{5.7}$$

and the product of the events "O occurs given X and λ" and "X occurs given λ"

give us the joint probability of O and X,

$$\mathbb{P}(O, X|\lambda) = \mathbb{P}(O|X, \lambda) \cdot \mathbb{P}(X|\lambda) \tag{5.8}$$

then the final probability of O is obtained summing over all the possible state sequences:

$$\begin{aligned} \mathbb{P}(o_1 o_2 \dots o_T|\lambda) &= \sum_{\{X|X \text{ is a feasible sequence of states}\}} \mathbb{P}(O|X, \lambda) \cdot \mathbb{P}(X|\lambda) \\ &= \sum_{X_1, X_2, \dots, X_T} \pi_{X_1} b_{X_1}(o_1) p_{X_1, X_2} b_{X_2}(o_2) \dots p_{X_{T-1}, X_T} b_{X_T}(o_T). \end{aligned} \tag{5.9}$$

The interpretation is quite simple: we start from the state X_1 with probability π_1 and we observe symbol o_1 with probability b_{X_1}. Then we move on state X_2 with probability p_{X_1,X_2} and we observe symbol o_2 with probability b_{X_2}, and so and so on. The computational cost of these operations is very high (about $2 \times T \times N^T$ calculations). Fortunately, there is a more efficient procedure called forward-backward procedure. We are going to explain how does the forward procedure work leaving the backward to the most curious readers. The forward procedure is based on the use of a "forward variable" $\alpha_t(i)$, defined as

$$\alpha_t(i) \coloneqq \mathbb{P}(o_1 o_2 \dots o_t, X_t = S_i|\lambda) \tag{5.10}$$

that corresponds to the probability of the partial observation sequence until time t and state S_i at time t. Then, by recursion

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) p_{i,j}\right] b_j(o_{t+1}) \qquad 1 \le t \le T-1,\ 1 \le j \le N, \tag{5.11}$$

where the initial and the final step are

$$\alpha_1(i) = \pi_i b_i(o_1)$$

and

$$\mathbb{P}(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) = \sum_{i=1}^{N} \mathbb{P}(o_1 o_2 \dots o_T, X_T = S_i|\lambda). \tag{5.12}$$

The recursive formula can be read as follow. Fixed a step t, $\alpha_t(i)$ indicates the probability that the sequence of observations occurs until time t and that in the last time the state reached is the S_i. So, in the next time $t+1$, the new forward variable is computed in two steps: firstly, summing α for all the possible ending states at the time t multiplied by the probability to pass in the new state S_j, and then multiplying the result for the probability of having the observation o_{t+1} in the state j. The computational cost of this approach is on the order of N^2T calculations.

The second question is much more "open", in the sense that there are several ways to answer it. The goal of this part is to find the optimal state sequence, according to the sequence of associated observations. The first approach we can have is to use as optimality criterion the individual most likely state, which means that we choose the state X_t that maximizes the likelihood of the relative observation. For this reason we define the probability of being in the state S_i at time t given O and the model as

$$\gamma_t(i) \coloneqq \mathbb{P}\left(X_t = S_i|O, \lambda\right) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}, \tag{5.13}$$

where $\beta_t(i)$ is the backward coefficient, which has the same meaning of $\alpha_t(i)$ except for the use of the last part of the sequence of observations (it involves $o_{t+1}o_{t+2}\dots o_T$). Obviously, summing over all the states S_i we observe that γ_t is a probability measure. As a result, we can use γ_t to solve our problem choosing X_t at the time t as

$$X_t = \underset{1\leq i\leq N}{\arg\max}\, \gamma_t(i) \qquad 1 \leq t \leq T. \tag{5.14}$$

The main issue related to this approach comes out when the HMM has one or more state transitions which have zero probability. For example if i_1 is the most likelihood state after i_0 but $p_{i_0,i_1} = 0$, then the choice of that state is not allowed by our chain and the method collapses. It's due to the fact that this solution doesn't

examine the probability of occurrence of the entire sequences of states. Since a basic solution could be to maximize the expected number of correct pairs of states (or triples of states and so on), the most used criterion is to find the single best state sequence (path). It is equivalent to maximize the probability $\mathbb{P}(X, O|\lambda)$. The algorithm for finding it is called *Viterbi algorithm* and it is based on dynamic programming. According to the Viterbi algorithm, the first step is to define the score along a single path at time t, taking into account the first t observations and ending in state S_i, as

$$\delta_t(i) \coloneqq \max_{X_1, X_2, \ldots, X_{t-1}} \mathbb{P}[X_1 X_2 \ldots X_t = S_i, o_1 o_2 \ldots o_t|\lambda]. \tag{5.15}$$

Similarly to the forward case above, by recursion we have

$$\delta_{t+1}(j) = \max_i [\delta_t(i) p_{i,j}] \cdot b_j(o_{t+1}). \tag{5.16}$$

It can be read as follow: given the best score at time t ($\delta_t(i)$), the best new one at time $t+1$ for the state j is the score of the most likely new state multiplied by the probability of having observation o_{t+1}. The approach based on dynamic programming is shown in algorithm 5. For doing it, we use two support arrays (T_1 and T_2, not to be confused with T, the length of the sequence).

The last issue is how to determine a method to adjust the model parameters in order to reach the maximum probability of the observation sequence given the model. The negative part of the theory is that there is no known optimal way to estimate the model parameters in order to reach maximize the probability of the observation sequence. On the other hand, we can find them such that $\mathbb{P}(O|\lambda)$ is locally maximized. Standard techniques for estimating HMM parameters involve batch learning, based either on specialized Expectation– Maximization (EM) techniques, such as the Baum-Welch (BW) algorithm, or on numerical optimization techniques, such as the Gradient Descent algorithm. We are going to have a breif look at one iterative procedure, based on Baum-Welch method. Let's define for first an important probability measure. We use $\xi_t(i, j)$ to indicate the probability of being in state S_i at time t and state S_j at time $t+1$, given the observation

input : V, the observation space,
S, the state space,
π, the initial probabilities,
$O = o_1 o_2 \dots o_T$, the sequence of observations,
P, the transition matrix,
B, the emission matrix;
output: $X = X_1 X_2 \dots X_T$, the most likely hidden state sequence.

for $j = 1;\ j \leq N;\ j++$ **do**
  $T_1[j, 1] \leftarrow \pi_j \cdot B_{j,o_1}$;
  $T_2[j, 1] \leftarrow 0$;
end
for $i = 2;\ j \leq T;\ i++$ **do**
  for $j = 1;\ j \leq N;\ j++$ **do**
    $T_1[j, i] \leftarrow \max_k(T_1[k, i-1] \cdot P_{k,j} \cdot B_{j,o_i})$;
    $T_2[j, i] \leftarrow \arg\max_k(T_1[k, i-1] \cdot B_{j,o_i})$;
  end
end
$z_T \leftarrow \arg\max_k(T_1[k, T])$;
$X_T \leftarrow S_{z_T}$;
for $i = T;\ i \geq 2;\ i--$ **do**
  $z_{i-1} \leftarrow T_2[z_i, 1]$;
  $X_{i-1} \leftarrow S_{z_{i-1}}$;
end
return X

Algorithm 5: Viterbi algorithm

sequence and the model:

$$\xi_t(i,j) \coloneqq \mathbb{P}(X_t = S_i, X_{t+1} = S_j | O, \lambda). \tag{5.17}$$

We can rewrite the probability measure above using forward and backward variables, so that

$$\xi_t(i,j) = \frac{\mathbb{P}(X_t = S_t, X_{t+1} = S_j, O|\lambda)}{\mathbb{P}(O|\lambda)} = \frac{\alpha_t(i) p_{i,j} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{i,j=1}^{N} \alpha_t(i) p_{i,j} b_j(o_{t+1}) \beta_{t+1}(j)}. \tag{5.18}$$

Recalling the previous definition of $\gamma_t(i) = \mathbb{P}\left(X_t = S_i | O, \lambda\right)$, we can observe that summing over all the states j the variable $\xi_t(i,j)$ we obtain exactly $\gamma_t(i)$, i.e.

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j). \tag{5.19}$$

Then, summing over the time index t individually both the two probabilities above, we obtain two meaningful interpretations

$$\begin{aligned} \sum_{t=1}^{T-1} \gamma_t(i) &= \text{ expected number of transitions from } S_i \\ \sum_{t=1}^{T-1} \xi_t(i,j) &= \text{ expected number of transitions from } S_i \text{ to } S_j. \end{aligned} \tag{5.20}$$

Finally, by using these two results in equations 5.20, we obtain a method for the estimation and the adjustment of the model parameters.

$$\begin{aligned} \bar{\pi}_i &= \text{ expected frequency in state } S_i \text{ at time } t = 1 \\ &= \gamma_1(i) \\ \bar{p}_{i,j} &= \frac{\text{expected number of transitions from } S_i \text{ to } S_j}{\text{expected number of transitions from state } S_i} \\ &= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \\ \bar{b}_j(k) &= \frac{\text{expected number of times in state } S_j \text{ observing } v_k}{\text{expected number of times in state } S_j} \end{aligned}$$

$$= \frac{\sum_{\{t\,:\,1\leq t\leq T,\ o_t = v_k\}} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

Through these three parameters we build a new HMM $\bar{\lambda} = (\bar{P}, \bar{B}, \bar{\pi})$ which has been proven that over estimates the likelihood, in the sense that $\mathbb{P}(O|\bar{\lambda}) > \mathbb{P}(O|\lambda)$. Proceeding iteratively and using $\bar{\lambda}$ instead of λ we improve the probability of O to be observed from the model until the method reaches a limit. At this point the limit is called maximum likelihood estimate of HMM and the method stops. It has been proved more accurately that maximizing the function

$$f(\lambda, \bar{\lambda}) = \sum_{X} \mathbb{P}(X|O, \lambda) \log[\mathbb{P}(O, X|\bar{\lambda})] \tag{5.21}$$

bring us to the three formulas above, in addition to the fact that its maximization leads to increased likelihood. The last comment about this procedure regards the fact that the stochastic constraints of the parameters (I mean, all the three parameters are probabilities and so they must sum up to 1) are satisfied at each iteration, which means that

$$\sum_{i=1}^{N} \bar{\pi}_i = 1, \quad \sum_{j=1}^{N} \bar{p}_{i,j} = 1, \quad \sum_{k=1}^{M} \bar{b}_j(k) = 1 \qquad 1 \leq i, j \leq N. \tag{5.22}$$

Finally, let's remark that since the problem has a structure typical for optimization problems, standard gradient techniques can be used to find the optimal values of the model parameters.

5.4 Why HMMs are suitable for DM1 Prediction

Generally, failure prediction can either rely on trends of faults or on their detection. About the first group, prediction methods mostly use continuously available measures such as workload to identify trends or anomalies. Indeed in this case the shorter is the time of reception of messages, the better will be the prediction. Methods belonging to the second group often only take the time of occurrence of the error into account. The first approach we use belongs to the second group. But

after the results, we decide to change method using one related to the first group. For the second attempt we make two assumptions, since we have to motivate the method. The first basic assumption in my approach is that the occurrence of failures can be predicted by identifying special patterns in the behaviour of the SPNs. This assumption is supported by the fact that dependencies among the parameters of the units and the DM1 exist, even if it is weak. Due to these dependencies, an error in one component of the unit should be caused by an anomaly trend of the correspondent selected SPN. The other not so restrictive assumption is that the new special faulty condition reached by a parameter always lead the unit to an error. It means that every time the system fall in the special faulty condition, then one DM1 inevitably occurs, no matter when exactly but before the system come back to a regular state.

Hidden Markov models (HMMs) in literature have been used successfully in pattern recognition tasks such as speech recognition or DNA sequence analysis. The main reason is that they exhibit the property to be flexible and adaptive while at the same time providing structure and simplicity that allows for formal analysis as well as deep understanding. The use of HMMs for DM1 prediction is further motivated by the fact that faults are unknown and can't be measured, but they produce an error message on their occurrences. This matches perfectly the theoretical notion of Hidden Markov models. In fact, there are unobservable hidden states (corresponding to trend patterns intervals leading to errors) producing symbols corresponding to errors.

5.5 Hidden Markov Models application

In this section we are going to build a Hidden Markov Model on our data. This is a significant probabilistic model which has as support idea the non-deterministic values of each SPNs. In fact, values are not only affected by noise, but also they are potentially spoiled during the transmission to the remote server. HMM is a doubly stochastic model which is appropriate for the problem since we can consider the unit consumption over time as the hidden stochastic process and the resultant

values of the SPNs as the measurable process.

Here, we provide two different kinds of application for HMM. In both these two applications we set the number of states of HMM to 10, which ideally corresponds to 10 intervals inside [min SPN100, max SPN100]. The first one is the common way that people use this algorithm. In this case the prediction of DM1s comprises two steps. First the model estimate the current state according to the previous events. Then, starting from the current state, it makes a prediction on future behaviour and, according to the next predicted state, the DM1 probability is computed in the following step.
The other application is quite more abstract and it is based on the idea that the model finds out a own state for all the faulty CAN messages. This is not certainly a sure assumption, since the model when training on data chooses the states as explained in the previous section. But it is supported by the strong mutation that SPN 100 and 100_*ma* may have in correspondence of the DM1 occurrence.

5.5.1 HMM for Failure Detection

The first application of HMM to DM1 prediction is divided into two steps, where at the start the model compute the probability δ_i^0 of being in a certain current state S_i after having encountered some observation $O = o_1 o_2 \dots o_T$. So, by the Viterbi algorithm it is easy to compute δ^0, since it is

$$\delta_i^0 = \max_{X_1, X_2, \dots, X_{T-1}} \mathbb{P}\left(X_1 X_2 \dots X_{T-1} X_T = S_i, o_1 o_2 \dots o_T | \lambda\right) \tag{5.23}$$

where $X_1 \dots X_{T-1}$ is the sequence of states encountered by the model to be in the state S_i at the time T, while the observation sequence is $o_1 o_2 \dots o_T$.
The goal of this approach is to compute the probability of a DM1 in the next step (so that it means in the next 10 minutes), which is the probability that the Markov process defined by the hidden states, transition matrix A and initial probability vector δ^0 produces an observation symbol with the DM1 at the following step, given that the current state is the S_i.
So, what we do here is to compute the emission probability of o_{T+1}, where o_{T+1}

contains the DM1. In order to do it, we assume to be currently in the state S_i, which is the most likely, and then we compute the probability by multiplying the probability to pass from state S_i to state S_j for the emission probability of o_{T+1} given the new state. Then we sum all these values over all the possible states and finally we obtain the probability of DM1 occurrence. Summarising,

$$\mathbb{P}(o_{T+1}|X_T = S_i, \lambda) = \sum_{i=1}^{N} p_{i,j} b_j(o_T). \tag{5.24}$$

Remark 7. Since the model training has a random component, in order to make more accurate results we repeat many times (20) training and test parts of the model. Each time we use 2 of the 3 units as training set and the last one as test set, by applying a sort of cross-validation (rotating the units). It means that finally we have done training and test of 60 models (3 times to choose the unit as test, 20 times to repeat all). As a result, we compute the average of the scores and we report all in a unique confusion matrix.

So, proceeding in this way we obtain the results shown in Figure 5.3, where the matrix has to be interpreted as a normalized one, with probabilities instead of absolute values.

The comment of this approach is quite obvious: since the results are very low, it means that it could not be the right model or it may be the wrong way to adapt it. In both these cases the results don't permit us trust its prediction. In a certain way, it might have been possible to predict the failure of this exact application because we are dealing with few DM1s. Since the amount of DM1s is very low, a model as HMM which works with probabilities and averages could fail because it isn't able to recognize the DM1 presence. So, for this reason the next approach should be better, as we don't consider the observations anymore but we provide a prediction base on hidden states.

5.5.2 HMM for faulty trend prediction

This subsection is related to a different application of HMM, without taking DM1 occurrences into account. The idea comes from the following question: Does HMM

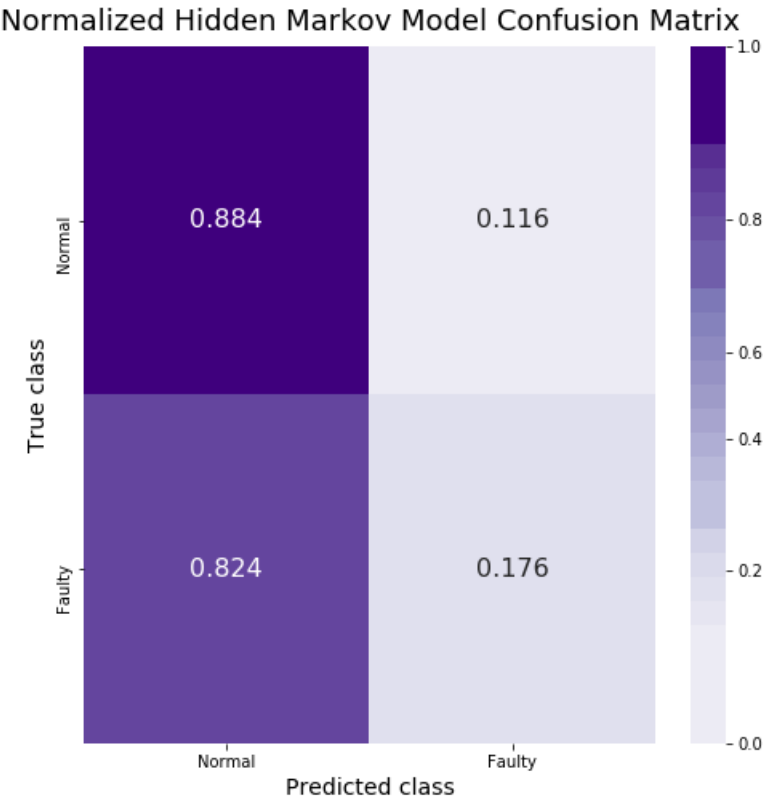


Figure 5.3: Confusion matrix showing the probabilities of predictions per classes. The sum of the rows is equal to 1, since, given the right class of an observation, the model predicts the two labels with complementary probabilities. The statistics are: *Precision*: 0.003, *Recall*: 0.176.

assign to each faulty CAN message the same states? It might be very surprising the answer, since we have no way to figure out what are the informations hidden in each state. In fact, states are configured and created according to the optimization problem we talked about in the previous section. But, at the same time, Hidden Markov Models are able to point out different patterns among data such that they can recognize if the time series is going into an anomalous trend. By the way, it is exactly what we are looking for: in order to make a prediction we have used the ploy to divide data into normal and faulty categories; however, using just states to look for the future trend doesn't need the label or to specify the closeness to DM1 occurrence. In Figure 5.4 it is shown the states prediction of HMM for a sample of 2000 points from the same unit. See that states are labeled taking sequences into account.

Defined the probability of being in the state S_i at time t given the sequence of observations O and the trained model as in 5.13,

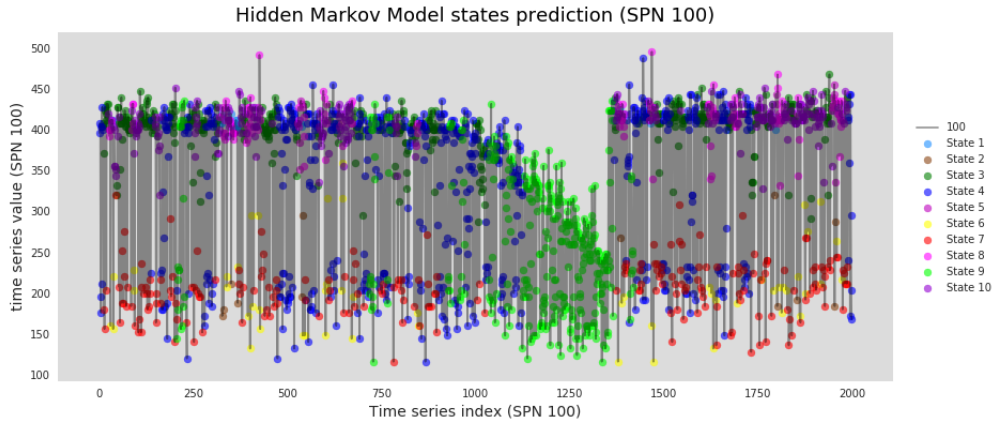
$$\gamma_t(i) = \mathbb{P}(X_t = S_i|O, \lambda) \tag{5.25}$$

we use this probability to find out the sequence of states which has the maximum probability,

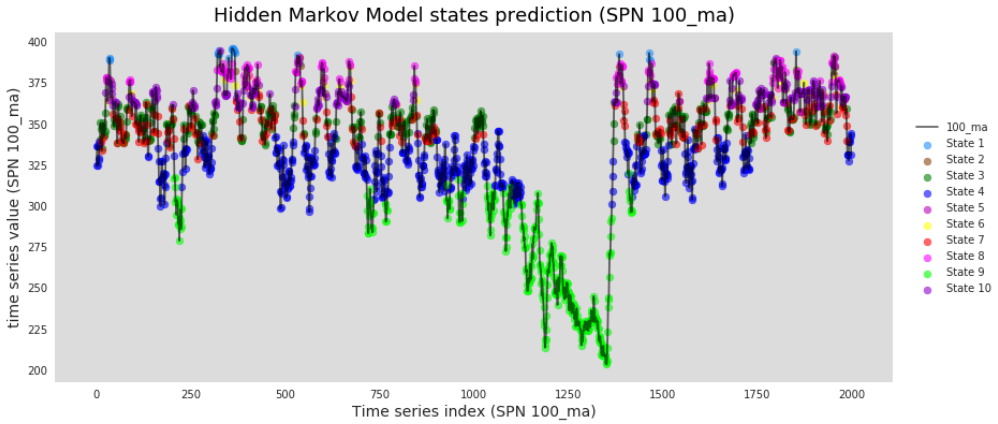
$$X_t = \underset{1\leq i\leq N}{\arg\max}\, \gamma_t(i), \tag{5.26}$$

for all $t \leq T$, where i indicates the state S_i. Note that the number of states is set to 10 and we make the assumption that one of this 10 is strictly connected to the faulty zone of CAN messages. Then, whenever HMM model enters into the faulty state, it means that its prediction is faulty. So we could create a confusion matrix, taking as true values the partition of data into normal/faulty according to the time label $30.0h$.

In the figure 5.5, we observe the results of the method: precision and recall are much higher than the previous part, even if we don't achieve so perfect scores. The main reason and the limit of this approach is related to the time label we set. Because of labeling data as faulty about 30 hours before one DM1 occurrence, the constraint is limiting the strength of HMM. In fact, as a result, through an



(a)



(b)

Figure 5.4: (a) SPN 100 time series with states predictions by Hidden Markov Model. For each state one color is used. (b) As for (a) but using SPN 100_*ma*. Observe that the state 9 is assigned to the faulty CAN messages, defining the faulty zone before that one DM1 happens.

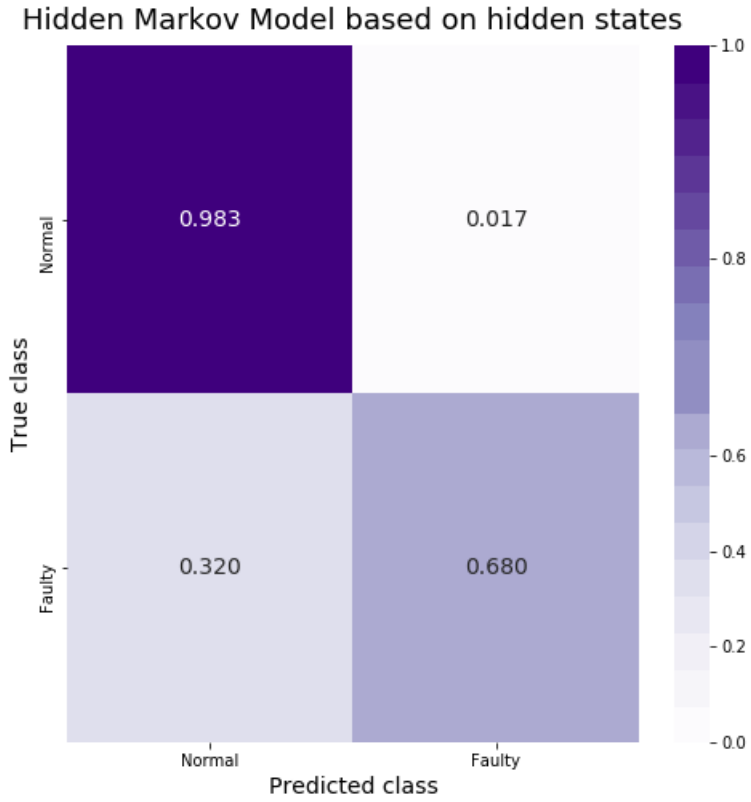


Figure 5.5: Confusion matrix showing the predictions per classes. The predicted label for a CAN message is "Faulty" if the Hidden Markov Model enters in the faulty state, "Normal" otherwise. In this case the statistics are:
- *Precision*: 0.630, - *Recall*: 0.680, - *AUC*: 0.831.

accurate empirical analysis of wrong predictions made by HMM, we might observe that each DM1 observation has preceded by a different duration of the mutation of values. So, increasing these scores are impossible, even changing time label because we should adapt a different time label to each DM1 occurrence, falling in the trivial overfitting case.

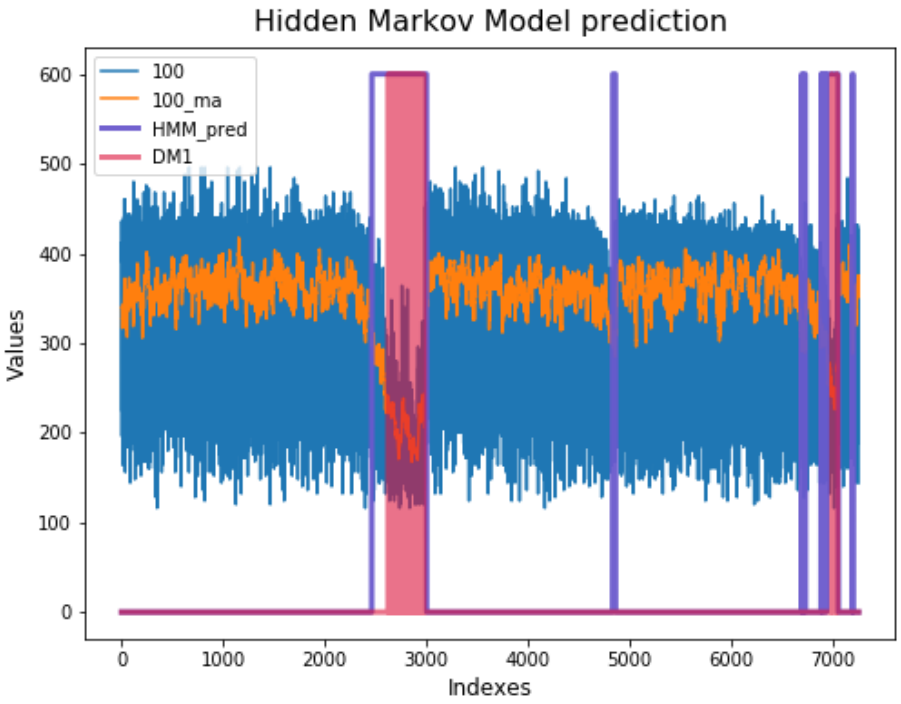


Figure 5.6: Hidden Markov Model prediction based on the state of anomalous trends. The red color is assigned to the DM1 messages, whereas the violet defines the interval containing CAN messages predicted as faulty by HMM.

As a support, look at the figure 5.6. It is shown an example of SPN 100 time series with its moving average, where DM1 occurrences are highlighted in red and HMM predictions in violet. We see that HMM is able to detect anomalous trend from the very beginning. In addition, it points out anomalies even if the DM1 doesn't occur. So, the model has a strong effect on trend anomaly detection, even thought it is a different problem with respect to the DM1 prediction.

Chapter 6

Prediction of Faulty CAN messages using Anomaly Detection techniques

6.1 Preliminary Concepts

In this short section we introduce two techniques used in the evaluation of the model and we explain the three score we look at for estimating the performances of each model. The first one is the stratified cross validation and it is used when the classes in the response variable are not balanced. The second technique is applied after the previous one and is called SMOTE. It is important to create data in order to balance the two classes. For the sake of honesty, we report the structure of SMOTE method even if we don't apply it to the anomaly detection problem.

Then we introduce the three performance metrics that are used to evaluate the models we will build: precision, recall and AUC. It is interesting that all these three scores are connected and can't be used separately. They in fact measure different qualities of the model and a high score of one of them doesn't imply that the model is perfect: by maximizing all of them we can find out the most suitable parameters for each model.

Keep in mind that all the graphs shown in this section are taken from the

Autoencoder model output, which will be explained in the section 6.5.

6.1.1 Stratified Cross validation

Generally, in supervised learning a classifier is constructed by using a learning algorithm which uses a labeled training set to learn from data. Then, its performance is usually measured by the accuracy, that is the probability of correctly classifying unseen cases, labeled by the trained classifier. A common strategy for the estimation of the true accuracy is to divide the starting dataset into two groups: training set and test set. The training set has the aim to train a classifier, whereas the test set is used to evaluate the accuracy of the chosen classifier. This process is usually repeated multiple times (with different random partitions of the data set into training and test sets), and the average of each performance gives an estimation of the true accuracy. This procedure is called cross validation.

However, what we want to do here is called stratified cross-validation [ZM00]. Fixed k, the number of cross validation times, the stratified cross validation divides the data set into k folds in such a way that a balanced distribution in the feature space is maintained for each class. It means that if we have two classes in the response variable and there are much more elements belonging to one than the other, then the division into training and test set is done holding the rate among the sizes of the classes. Suppose, as in our case, to have 77261 data and 75678 of these are labeled with 0, whereas just 1583 data are in the class 1. Assume that the cross validation divides data into $k = 5$ folds, that is 61808 data inside the training set and 15453 inside the test set. So, if the cross-validation is stratified, then in every fold the training set will be composted by exactly 1266 elements of the class 1 and 60542 of the class 0, while the test set by 317 elements of the 1 and 15136 of 0. This numbers come from the fact that the rate 1266/61808 and 317/15453 has to be equal to the initial rate $1583/77261 \approx 2.05\%$. It is easier to think that each fold of 15453 elements has to hold the rate 0.0205 among the two classes and so it is composted as above. The normal common cross-validation, instead, choose the data without taking this proportion into account and so it could be possible that one or more partitions contains only elements belonging to the same class.

This usually lead the model to wrong results or, eventually, to formal errors. The comparison between stratified and normal cross-validation is shown in picture

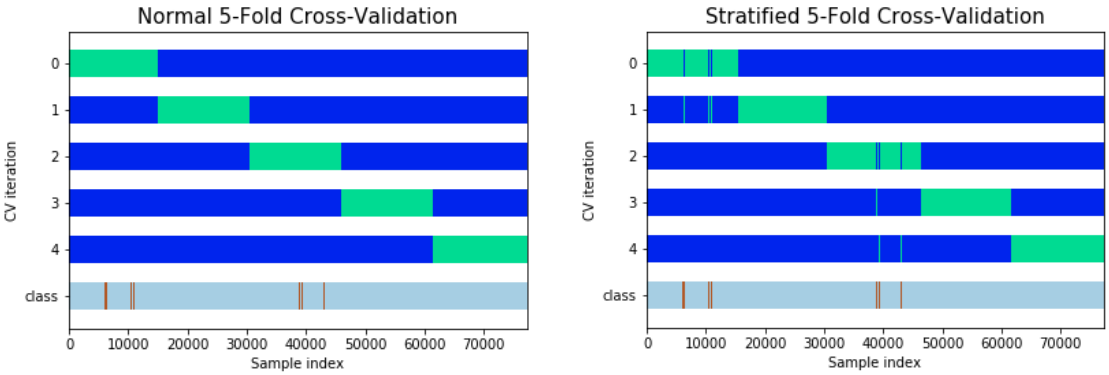


Figure 6.1: Normal and Stratified 5-Fold cross-validation using 30.0h as time label gap. The green lines represent data inside the test set, while dark blue lines represents the training set. At the bottom, after the CV iterations, the general partition of the entire dataset in faulty and normal classes is shown.

6.1.2 SMOTE: Synthetic Minority Over-sampling Technique

Synthetic Minority Over-sampling Technique (SMOTE) is an approach to the construction of classifiers from imbalanced datasets [Cha+02]. As we can imagine, a dataset is defined as imbalanced if the number of elements in each category is not approximately the same. This is a common problem, since often in the real world datasets are generally made of "normal" data with only a small percentage of "abnormal" samples. In addition, usually the cost of misclassifying an abnormal data as a normal one is much higher than the cost of the reverse error. So, the under-sampling technique of the majority class is useful to increase the sensitivity of the classifier to the minority class. In fact, through balanced classes the classifier should be able to learn how to distinguish them. However, decreasing the number of data inside the bigger class makes the dataset be much smaller and it could be seen as a waste of data. This might be a problem mostly for machine learning algorithm which needs of a lot of data to be very accurate. So, a substitute method

is the over-sampling by creating new synthetic samples instead of using a sort of replacement. In order to generate new data points belonging to the smaller class, the method operates in the feature space. It selects for any points x_i its k nearest neighbours (usually $k = 5$, but it depends on the number of samples we need) and for each of them it create another point by choosing randomly it inside the linear combination set of the previous two points. In practice, we consider two points in the smaller class, say x_i and x_j $(i \neq j)$ and then we create the third data point p by selecting a random number $\alpha \in [0, 1]$ and

$$p = \alpha x_i + (1 - \alpha)x_j.$$

This forces the decision region of minority class to become more general.

It seems to be feasible for out problem which has very unbalanced classes. On the other hand, SMOTE can't be applied in all the situations since it depends on the distribution of data. In fact, if data a not normal distribution like categorical variables or simply bi-modal variables, the interpolation and the linear combination of 2 points belonging to different classes (for the first case) or close to different peak (for the second one) could produce a third data point that is not very probable according to the distribution. In the first case we can't create another class, whereas in the second one we will have a point in a low density zone and it won't be so useful. But after the feature selection, the CAN parameters 100 has the smaller class with a distribution very similar to the normal one (even if it's not normal) and so SMOTE is able to generate synthetic data without changing a lot the initial distribution of values. To be confident with this assumption, look at the figure 6.2 and in particular at the faulty CAN values.

This kind of approach is really useful in mostly of the problems with unbalanced classes. However, since we adapted the predictive maintenance structure to the anomaly detection one, we prefer not to apply it on the algorithm. The reason is related to the fact that we want the algorithm to recognize if a data point is anomalous or normal. So, for this reason we cannot have a lot of anomalous points, especially when we use distance based anomaly detection methods. Since for the methods that aren't related to anomaly detection strategies it results to be very

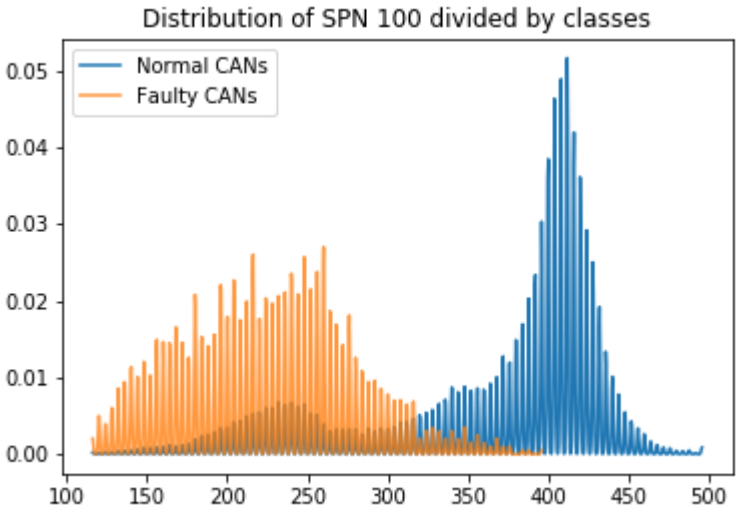


Figure 6.2: Distribution of SPN 100 obtained by separating values belonging to the normal class from the others belonging to the faulty class.

strong and feasible, we decide to report it as a potential attractive alternative.

6.1.3 Precision, Recall and AUC statistics

Precision and Recall statistics are typically used in binary classification problems in order to study the output of a binary classifier. They are useful measures of success of prediction when the classes are very imbalanced. Generally speaking, precision score is a measure of the relevance of the result, while the recall is a measure of how many truly relevant results are labeled. We indicate the two classes with 0 and 1, where of course 1 represents the class with rare events (faulty CAN).

The Precision-Recall curve shows the trade off between precision and recall. An example it is reported in Figure 6.3. A high area under the curve represents both high recall and high precision. In general high precision values relate to a low false positive rate, whereas high recall values relate to a low false negative rate. As a result, high scores for both show that the classifier is returning very precise results (high precision), as well as returning a majority of all positive results (high recall). On the other hand, a model with high recall but low precision

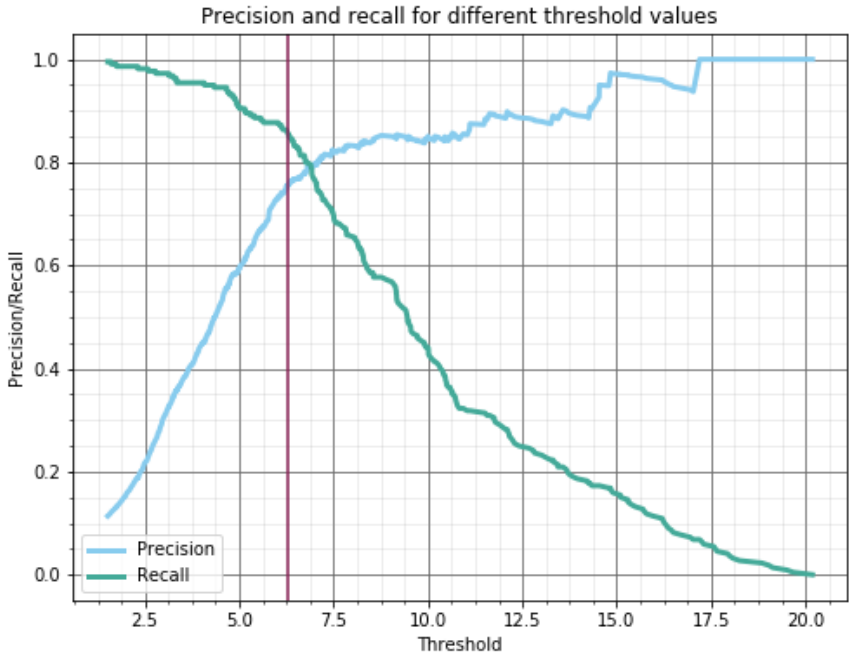


Figure 6.3: Precision and Recall curve by Autoencoder model. In red is shown the best F_1 score value which maximizes both precision and recall scores.

returns many 1 values, but most of them are incorrect when compared to the real labels. It implies that the model predict many 0 values as 1 and for this reason the precision is low. On the contrary, a system with high precision but low recall is just the opposite, returning very few 1 labels, but most of its predicted labels are correct when compared to the real labels. An ideal system with high precision and high recall will return many results, with all results labeled correctly.
By looking at the confusion matrix, through the 4 values reported inside it, we can define and compute the precision, recall and AUC scores. An example of confusion matrix, illustrating the positions of TP, TN, FP and FN, is shown in Figure 6.4. Let's remember that true positives / true negatives are data point classified respectively as positive / negative by the model that actually are positive / negative, that means the model assign to these data the correct label. On the other hand, false negatives are data points that the model predicts as negative while actually are positive (incorrect) and false positives are cases the model incorrectly labels as positive that are actually negative.

Definition 6.1. The *precision score* P is defined as the number of true positives T_p over the number of true positives plus the number of false positives F_p,

$$P \coloneqq \frac{T_p}{T_p + F_p}. \tag{6.1}$$

Definition 6.2. *Recall* R is defined as the number of true positives T_p over the number of true positives plus the number of false negatives F_n,

$$R \coloneqq \frac{T_p}{T_p + F_n}. \tag{6.2}$$

Definition 6.3. Both precision and recall are related to the F_1 *score*, which is defined as the harmonic mean of them,

$$F_1 \coloneqq 2 \times \frac{P \times R}{P + R}. \tag{6.3}$$

Note that the precision may not decrease with recall. The definition of precision 6.1 shows that decreasing the threshold of a model classifier may increase the denominator, by increasing the number of results returned. However, if the threshold was previously set too high, the new results may all be true positives, which will increase precision. If the previous threshold was about right or too low, further lowering the threshold will introduce false positives, decreasing precision. Recall is defined as $\left(\frac{T_p}{T_p+F_n}\right)$, where the denominator does not depend on the classifier threshold. This means that decreasing the classifier threshold may increase recall, because it increases the number of true positive results. It is also possible that lowering the threshold may leave recall unchanged, while the precision fluctuates. The relationship between recall and precision can be observed in the precision-recall plot where for each threshold corresponds a precision and recall value. Let's summarize:

- Recall: ability of a classification model to identify all relevant instances;
- Precision: ability of a classification model to return only relevant instances;
- F_1 score: single metric that combines recall and precision (harmonic mean).

Usually to find out the optimal threshold we maximize the F_1 score: since it is a statistic which penalizes extreme values of recall and precision, by maximizing it we should find the best threshold to have both precision and recall high at the same time.

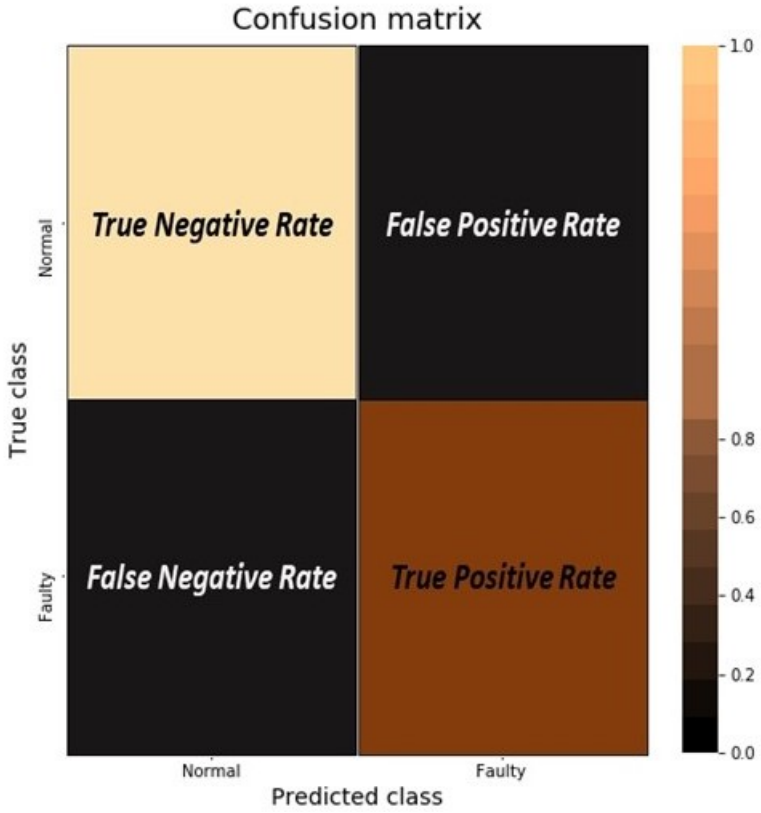


Figure 6.4: Confusion matrix showing true positives, true negatives, false positives and false negatives positions. Colors reflect target ranges.

ROC curve and AUC score

The other main visualization technique for showing the performance of our classification model is the Receiver Operating Characteristic (ROC) curve. The idea is relatively simple: the ROC curve shows how the recall vs precision relationship changes as we vary the threshold for identifying a positive in our model. The threshold represents the value above which a data point is considered in the positive class. Since we have a model for identifying a faulty CAN, our model might

output a score for each CAN parameter between 0 and 1 and we can set a threshold in this range for labeling a timestamp as having the fault (a positive label). By altering the threshold, we can try to achieve the right precision vs recall balance.

A ROC curve plots the true positive rate on the y-axis versus the false positive rate on the x-axis. The true positive rate (TPR) is the recall and the false positive rate (FPR) is the probability of a false alarm. Both of these can be calculated from the confusion matrix:

$$TPR = \frac{TP}{TP + FN}; \tag{6.4}$$

$$FPR = \frac{FP}{FP + TN}. \tag{6.5}$$

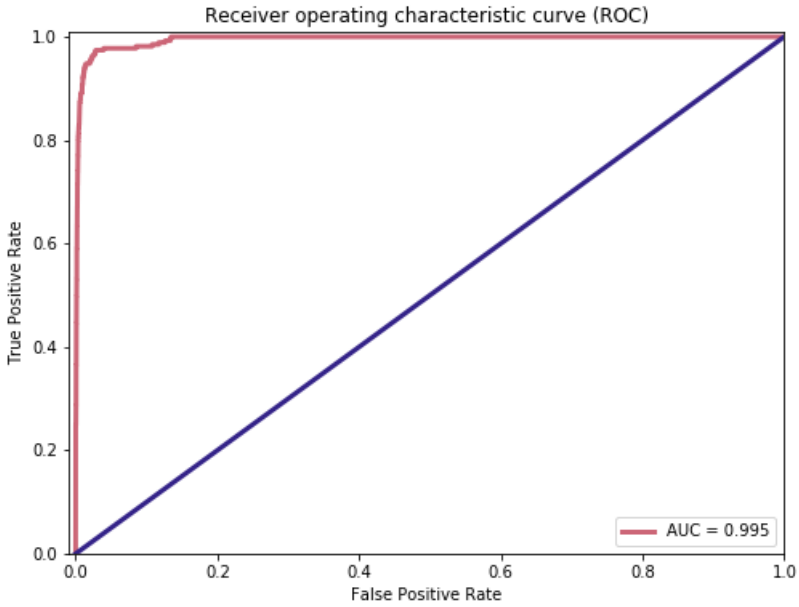


Figure 6.5: ROC curve for the predictive model Autoencoder with AUC score.

Look at the figure 6.5. The dark blue diagonal line indicates a random classifier and the red curve relates the classification model. Obviously for a given model, we can only stay on one curve, but we can move along the curve by changing our threshold for labeling a positive case. Generally, as we decrease the threshold, we should move to the right and upwards along the curve. On the other hand, with a threshold of 1.0 we would be in the lower left of the graph because we identify

no data points as positives leading to no true positives and no false positives ($TPR = FPR = 0$). As we decrease the threshold, we label more data points as positive, leading to more true positives, but also more false positives (the TPR and FPR increase). Eventually, at a threshold of 0.0 we identify all data points as positive and find ourselves in the upper right corner of the ROC curve ($TPR = FPR = 1.0$).

Finally, we can measure a model's ROC curve by computing the total Area Under the Curve (AUC), which is a metric that falls between 0 and 1. As most of statistical scores, a higher number indicates a better classification performance. In the graph 6.5, the AUC for the red curve will be greater than that for the blue line, meaning the red model is better at reaching a composition of precision and recall. A random classifier (the dark blue line) achieves an AUC of 0.5 and it means that it is impossible (in the sense that it will have no meaning) for a model to have an AUC score less than 0.5.

6.2 Why Anomaly Detection for Predictive Maintenance

Every machine learning application will encounter in future many different kinds of risks such as statistical uncertainties. This will happen inevitably, since nature of data is probabilistic and not deterministic. One such risk is the presence of statistical errors or inconsistencies due to distributional outliers or noisy observations. Precisely because of this case, anomaly/novelty detection was born. It highlights some of these risks, and its development is one of the most important and critical machine learning issue. The idea to apply Anomaly Detection methods to the problem comes from articles [GE18; PAD18], even if their approach is completely different from the one we use.

In the classic anomaly detection problem, we have data from a "normal" class of values, emerging from some unknown distribution, and the goal is to build an appropriate classifier which is able to detect out of distribution "abnormal" values. There are a few variants of this basic anomaly detection problem. For example, in

the positive version, where the outliers are labeled as 1, we have a sample from the "normal" class, as well as a sample that is contaminated with abnormal instances. All this data are obviously passed as input without their label.
So, anomaly detection is the problem of identifying whether a new data point has to be considered as an inlier or an outlier. From a pure statistical point of view this process usually occurs while the distribution of inliers is the only available information. This is also the most difficult and significant situation because outliers are often very rare and sparse. They could be even dangerous to experience and for this reason we have to rely only on inlier training data. The most used approaches tend to either use a one-class classifier, or to use as anomaly score, the reconstruction error of the chosen machine learning algorithm.

An important clarification has to be done and concerns the role of anomaly detection for this work, which is classified as predictive maintenance. These two concepts seem to be in contrast to each other. In fact predictive maintenance is, somehow, a set of techniques having the aim to predict and prevent a fault in the system a few hours in advance, while anomaly detection consist of recognizing an outlier after it occurs. So, the first has to anticipate the fault and to prevent it, while the second one has to recognize it afterwards. Now the questions are clear:

- How can we combine them together?
- Why their intersection could be useful for our issue?

Basically, we want to predict in advance when a DM1 occurs. But it is caused by a significant change in some CAN parameters and, hopefully, this mutation inside the CAN values starts many hours before the failure. So, for this reason, we have built the dataset dividing CAN messages into faulty and normal, since it wants to mean that all the values close to the DM1 occurrence are very different from those which are far from it. In this case the enlightening idea is that we can adapt the problem into one of anomaly detection, where the anomaly is not the DM1 occurrence but it corresponds to the faulty CAN message. So, anomaly detection techniques are applied to recognize the faulty CAN message in order to understand if the vehicle is in a phase close to the DM1 failure. Then, after having

decided if a CAN is faulty, the maintenance could be done in advance since we know that the first DM1 will appear within few hours (depending on the time label gap chosen). The second question deals with a more technical aspect. DM1 occurrences are rare and sparse events and they correspond to about 1.5% of the data. Common machine learning techniques will make random results, as they will not be able to point them out. However, anomaly detection deals with this kind of events and for this reason it produce more accurate results. In the remaining part of this work, we are going to apply different anomaly detection principles to detect faulty CAN messages and to prevent the DM1 occurrences.

6.3 K Nearest Neighbor Outlier Detection

In the previous section 6.2 we said that the classical methods of anomaly detection use a classifier connected to the reconstruction error of data. The main problem with this approach is that in many situations, we might simply not have enough knowledge about the underlying data distribution to set the problem in that way. So, here is presented an uncommon method based on the distance between data. All the procedure is supported and founded by the sentence [RRS00]:
An object O in a dataset T is a $DB(p, d)$-outlier if at least fraction p of the objects in T lies greater than distance d from O.
The term "$DB(p, d)$-outlier" is a shorter notation for a Distance-Based outlier (detected using parameters p and d). It means that a point O in a dataset is a distance based outlier with respect to parameters p and d if no more than $p \times size(T)$ points in the dataset are at a distance of d or less from O.

This definition of outliers has the pros to be very intuitive and easy to understand, as well as to have a low computational cost even for very large datasets. However, it has two cons:

- We need to specify a distance d which could be difficult to determine and could make the problem become much more computationally heavy.
- It doesn't build a ranking for the outliers, in the sense that if a point has many neighbours with respect to the distance d, then it is more probable

that it is considered as inlier or, in somehow, a weaker outlier then a point which has few neighbouring points.

The algorithm based on distance outlier definition that we use is the kNN. Fixed the l^2 distance as a measure of how much data points are different from each others, the method takes as score for each data point its distance from the k-th nearest neighbour. In fact, fixed the number of neighbours k and a point x, $D^k(x)$ indicates the l^2 distance of x from the k-th closest point and it is a measure of how much x is an outlier for the dataset. The support idea of this approach is that an outlier is more isolated than the normal points, since it is a rare and sparse event. Here we immediately note that the choice of d is extremely significant because it is the anomaly score used for the classification.

Definition 6.4. Given N data points $\{x_1, \ldots, x_N\}$ and fixed two parameters n and k, a point x_i is a D_n^k outlier if there are no more than $n-1$ other points x_j such that $D^k(x_j) > D^k(x_i)$.

In this way we can make a ranking according to the D^k distance of each data point. So, the anomaly score is calculated for each data point. Then, by using just the training set, the algorithm select a threshold to separate the two classes (inlier and outlier). In order to do it, as it will be explained much better in the next section, the model requires as parameter an approximate number which indicates the amount of expected outliers (more or less, usually they set a rate of the entire dataset as 1% or 2%). So, through the knowledge of the number of outliers, the algorithm looks for the distance value (the anomaly score) of the last point in the ranking sorted by anomaly score and the threshold is set to its value. So, once the threshold is chosen, a data point in the test set is claimed to be outlier if its anomaly score exceeds the threshold. The result of the method will be a classification of all the data set points into inliers or outliers.
Another consideration has to be made: to compute the kNN distance, the algorithm applies the $K-d$ *Tree Algorithm*, which is a common approach to approximate kNN based on spatial clustering. The $k-d$ tree is a binary tree in which every node is a $k-$dimensional point. Every node of the tree, which is not a leaf,

could be thought as if it implicitly generates an hyperplane partitioning the space into two different parts known as half spaces. Then the representation is quite simple and immediate: all the data belonging to the left part of the hyperplane are put together as representing the left subtree of that node, whereas the other points to the right represents the right subtree. To have more details about how $K-d$ Tree Algorithm works you can read the article [Ota+13].

Now we can apply the algorithm to our situation. It is important to note that we have to choose the number of neighbours of each point. Since our dataframe has more than 77.000 rows, it has no meaning to select a small number of neighbours. For this reason, our choice is to put $k = 1000$.
For the validation part, we use the cross validation stratified (without the SMOTE technique), as explained in the section 6.1.1.

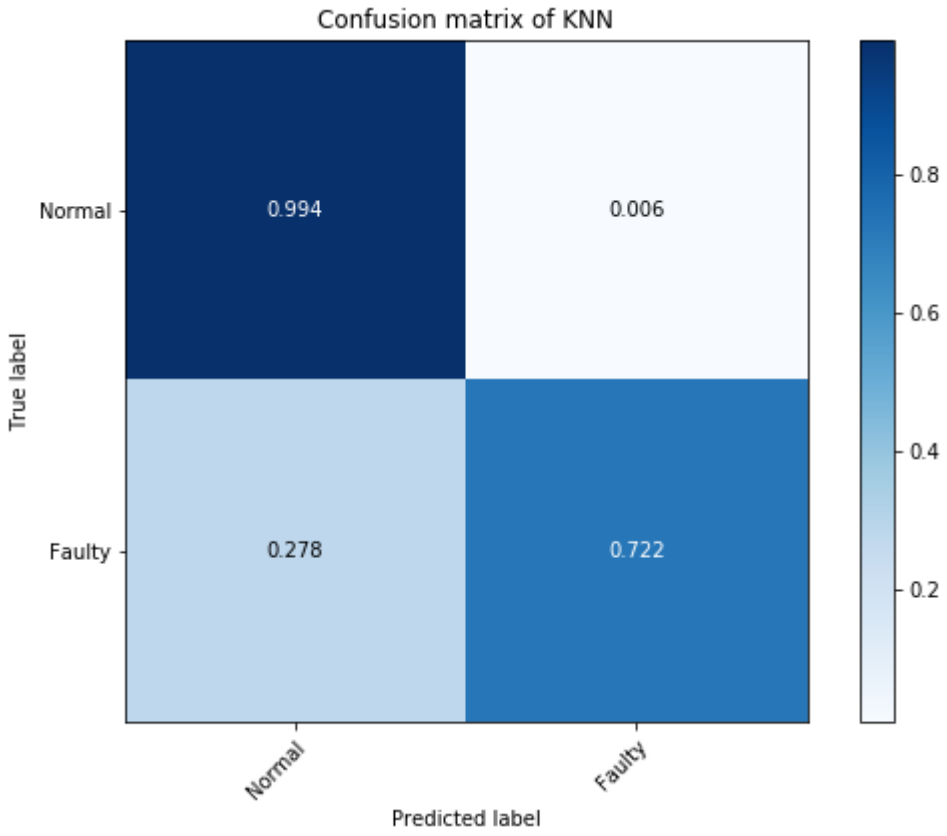


Figure 6.6: Confusion matrix for KNN outlier detection with statistics:
- *Precision*: 0.710 - *Recall*: 0.722 - *AUC*: 0.858.

In the Figure 6.6 it is shown the confusion matrix related to one fold of cross-validation. The results are not as perfect as we would want since the recall and precision are not so high. In fact, probably the method tends to predict faulty for all the data located at the boundary of the dense zone, while it predicts normal if a point fall inside the crowded region, as we can see in the Figure 6.7.

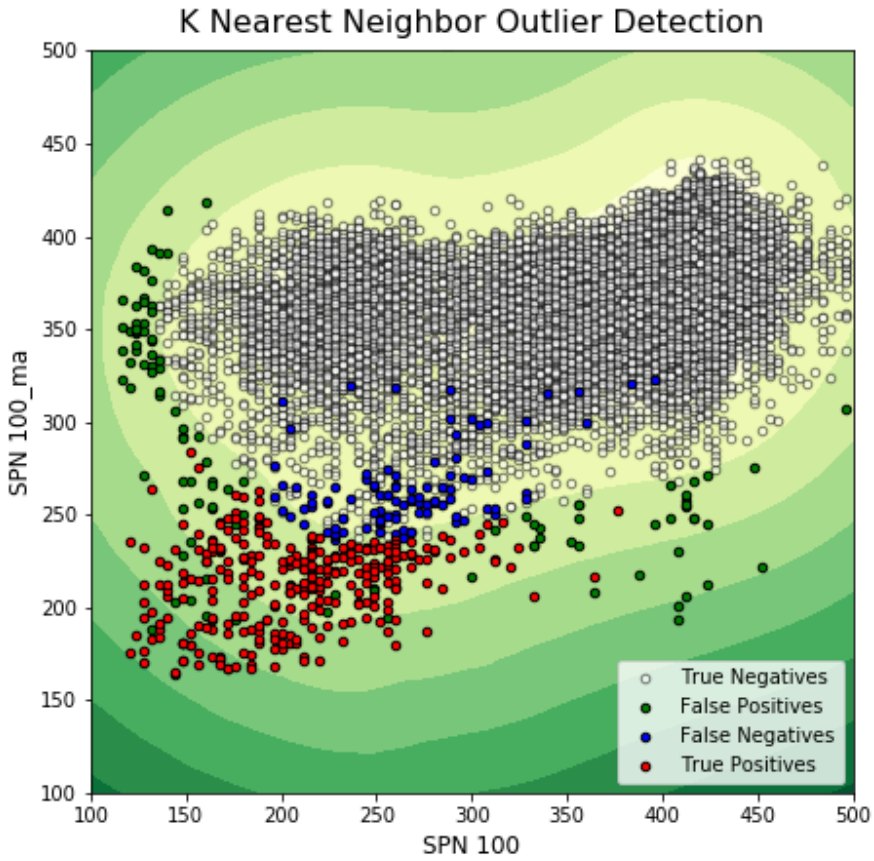


Figure 6.7: Scatter plot related to kNN Outlier Detection showing for each data point its model prediction. The background indicates the scores of a potential point in that area: the darker the color, the higher the anomaly score. In this case the dataset is divided into training and test (stratified) with rate 0.25.
The statistics are: - *Precision*: 0.724, - *Recall*: 0.747.

So, as a result, the model works very accurately with data located in the dense

area, whereas it tends to predict faulty too many times when only one of the two features is low. In this case the point might be in very close to be considered as faulty, such as values slightly earlier in time than the beginning of the faulty class. On the other hand, points with low moving average values and normal engine oil pressure could be labeled as faulty because of the first feature even if they might be the first CAN messages after the maintenance. So, with respect to the limits of the data, we can conclude by saying that the results obtained with the kNN Outlier Detection method are really competitive. After cross-validation technique, by averaging the statistical scores, we obtain the final result about a general accuracy of the model which provides:

$$Precision : 0.706, \quad Recall : 0.702, \quad AUC : 0.848.$$

6.4 Isolation-based Anomaly Detection: Isolation Forest

The method we propose in this section is called *Isolation Forest* and it is a tree based method [LTZ08; LTZ12]. It consist on creating many trees which, by splitting data in the canonical way, produce anomaly scores according to the average path lengths. Basically, a tree splits one data point as many times as it is isolated from all the others. Then, the support idea is that a point is anomalous if it needs few splits to be isolated, since it has values far from the normal point ones. The approach is interesting because it isn't based on distance or density measures, so that it has low computational cost. In addition, it is able to handle large data sets both in terms of large amounts of data and in terms of high dimensional points. By building randomly binary trees, each instance is recursively partitioned and the paths connected with anomalies are significantly shorter because of two reasons:

- Anomalous data points occupy less-dense regions so that trees need few iterations to completely isolate them.
- If a data has at least a very different attribute, it is more likely to be separated

early with respect to points with common values.

As we said above, many different trees are generated and each one creates randomly many splits so that each point has a lot of paths. By computing the average path lengths, we find out the true path score that will be normalized in order to compare path lengths from models with different sub-sampling sizes.
So, now let's give two definitions:

Definition 6.5. A tree is an *Isolation Tree* if its nodes are either external-nodes with no child or internal-nodes with exactly two offspring (T_l, T_r) and a test. The test is composted by an attribute q and a split value p such that the test $q < p$ makes a data point go into T_l or T_r.

Definition 6.6. Given a data point $x \in \mathcal{X} = \{x_1, \ldots, x_n\}$, the *path length* $h(x)$ is measured by the number of edges crossed by the point x, from the root node to its own external node.

Given the dataset $\mathcal{X}$, let $X \subset \mathcal{X}$ be a sample of data. Then, we use X to generate the isolation tree in order not to fall into an overfitting problem. The procedure is quite simple and common:

- Extract X from $\mathcal{X}$;
- Recursively divide X by selecting the attribute q and splitting the value p randomly;
- If a data fall into an external node, then stop;
- Otherwise, stop the algorithm when all data at the node have the same values.

After a brief introduction to isolation trees and about the general idea of the algorithm, let's now pass to the next question: how to use isolation trees for anomaly detection?
The first part is related to the training stage, where isolation trees are generated by recursively partitioning a sub-sample X until all the data points fall into an

external node. The choice of sub-sample X is made without replacement and, obviously, the subset is randomly selected by fixing just its size M (so that $|X| = M$). In algorithm 6 it is reported the pseudo code of the construction of each isolation tree.

input : X, the input randomly selected sub-sampling of data
output: I, the isolation tree.

if X *can't be divided anymore* **then**
  return *externalNode*{Size $\leftarrow |X|$};
end
else
  let Q be a list of attributes in X;
  randomly select an attribute $q \in Q$;
  randomly select a split point $p \in [\min q, \max q]$;
  $X_l \leftarrow$ *filter*$(X, q < p)$;
  $X_r \leftarrow$ *filter*$(X, q \geq p)$;
  return *internalNode*{
    Left $\leftarrow$ isolationTree(X_l),
    Right $\leftarrow$ isolationTree(X_r),
    SplitAttribute $\leftarrow q$,
    SplitValue $\leftarrow p$}
end

Algorithm 6: IsolationTree

Note that it is a recursive algorithm, since if a node is an internal node, than for each of its two offspring we must repeat the algorithm. It will stop whenever each data has its external node. Of course, at each step we save even the split attribute, which corresponds to the feature that splits data, and the splitting point, that is randomly chosen inside the maximum interval.
Two parameters as to be set for generating the correct model: the sub-sampling size M and the number of trees T. Since the problem has a lot of data, our choice is to set the size M of X to the 0.15% of the entire dimension of the dataset. This is related to the decision to use 1000 trees, which is a high number. So, in order to avoid overfitting, we don't use a high percentage of data. Furthermore, the algorithm is much more efficient and accurate when setting a low size of the sample. This consideration not only has theoretical supports, but also by providing

empirical scores trying as many different values as we can, we can observe that in practice it is much better to put scores to the order of $0.01 - 0.5\%$.

input : x, the data point,
I, the isolation tree,
$hlim$, the height limit,
e, the current path length.
output: P, the path length of x.

if *I is an external node or $e \geq hlim$* **then**
  return e + cost(I.size);
end
$a \leftarrow I$.splitAttribute;
if $x_a < I.splitValue$ **then**
  return *ComputePathLength*(x, I.left, $hlim$, $e + 1$);
end
else
  return *ComputePathLength*(x, I.right, $hlim$, $e + 1$);
end

Algorithm 7: ComputePathLength

The evaluation step, reported in algorithm 7, has as output value the sum of the path length and a cost adjustment. So first, the algorithm has to compute the path length of a data point x, counting the amount of edges of the tree x pass through. It is of course an iterative algorithm, since at each step the data attribute is splitted according to the random split selected. Then it goes on until an external node of the isolation tree is reached by x. The second part of the sum is an adjustment, based on the size of the external node which means on how many data points are inside it. In fact we would want to take the size of external nodes into account in order to create a score which is comparable within nodes of different sizes. So, called M_I the size of the isolation tree, we define the cost function as

$$cost(M_I) := \begin{cases} 2H(M_I - 1) - 2 \cdot \frac{M_I - 1}{n} & \text{if } M_I > 2; \\ 1 & \text{if } M_I = 2; \\ 0 & \text{otherwise.} \end{cases} \tag{6.6}$$

The cost function definition involves H, which is the harmonic number,

$$H(i) \coloneqq \sum_{k=1}^{i} \frac{1}{k} \approx \ln(i) + 0.5772156649. \tag{6.7}$$

The explanation is not so trivial. Since all the procedure depends on the amount of data we put in the randomly selected subset X, be careful to observe that the number or data inside an external node increases with the order of $\log |X|$. In fact there are many exponential splits to reach an external node. On the other hand, the maximum possible height of an isolation tree grows in the order of $|X|$, because a tree might separate at each split just one point from all the others. So, an adjustment is needed to compare different isolation trees cost. The use of harmonic numbers comes from the Binary Search Tree method.
Finally we can compute the anomaly score of a data point x as

$$s(x, M_I) \coloneqq 2^{-\frac{E[h(x)]}{cost(M_I)}}, \tag{6.8}$$

where $E[h(x)]$ stands for the average of $h(x)$ (the length of a path). The score has 3 properties:

- if $E[h(x)] \to 0$, then $s \to 1$; it follows the idea that an anomalous point has a low number of splits and so low $h(x)$;
- if $E[h(x)] \to M_I - 1$, then $s \to 0$; on the contrary, if the number of split is very high (to the maximum of $M_I - 1$) then the score s goes to zero;
- $E[h(x)] \to cost(M_I)$, then $s \to 0.5$; it means that we can't distinguish accurately if it is an anomalous point.

Informally speaking, the cost function is the theoretical average path length and if the expected path length ($E[h(x)]$) is equal to it, then it means that we can't say surely what kind of point it is.

Currently, we point out how anomaly scores are computed, in order to make a classification which separates inlier from outlier (normal data from anomalous data). However, it is not clear the way the algorithm operates: how it is computed

the threshold to make a correct classification? What's the role of the training set? How do we test the model?

To answer this questions we will not go into the mathematical details, but we provide just an accurate explanation. At the beginning, by using training data, the algorithm calculates for each data point one anomaly score value as we explained before. Then it has to be chosen the right threshold, obviously through the training anomaly scores. The threshold is based on the concept of "contamination". Contamination K is an a-priori knowledge and it corresponds to the percentage of outlier with respect to all the data points and it can be estimated through the training set. In our case, fixed the training set, we put the contamination value as the empirical number of outliers in the dataset X, so that

$$K(X) = \frac{\text{Number of anomalous data in } X}{\text{Total number of data in } X}. \tag{6.9}$$

In order to find out the right threshold, we consider the total number of anomalous data. Then, the value of the threshold is set to the last score in the ranking with exactly $|X| \times K(X)$ most abnormal data. The threshold is calculated to classify data in the test set. So, the ending part is related to testing the method on the test set. As you can imagine, for each data point in the test set the model computes its anomaly score and, according to the previous threshold, it makes the classification claiming anomalous data (which is label 1) if its score is such that $s \geq threshold$, normal point otherwise (label 0). In the figure 6.8, it is shown the results obtained by applying isolation forest to our dataset splitted with rate 0.25 in training and test set with the stratified technique.

Results are clearly very high for a simple and efficient model as it is. Since it uses a feature at the time for the splitting part, it is unsurprising that the errors it makes are in the intermediate position between normal values and faulty ones. As a result, the main disadvantage of this approach is that features have the same weight. Looking carefully at the graph 6.8, we can observe that green points are located into the same zone at the left, whereas the blue points are all in the right one. It means that the model predicts too many times faulty when the x-axis is low, so when SPN 100 is too low. At the same time it should predict faulty when

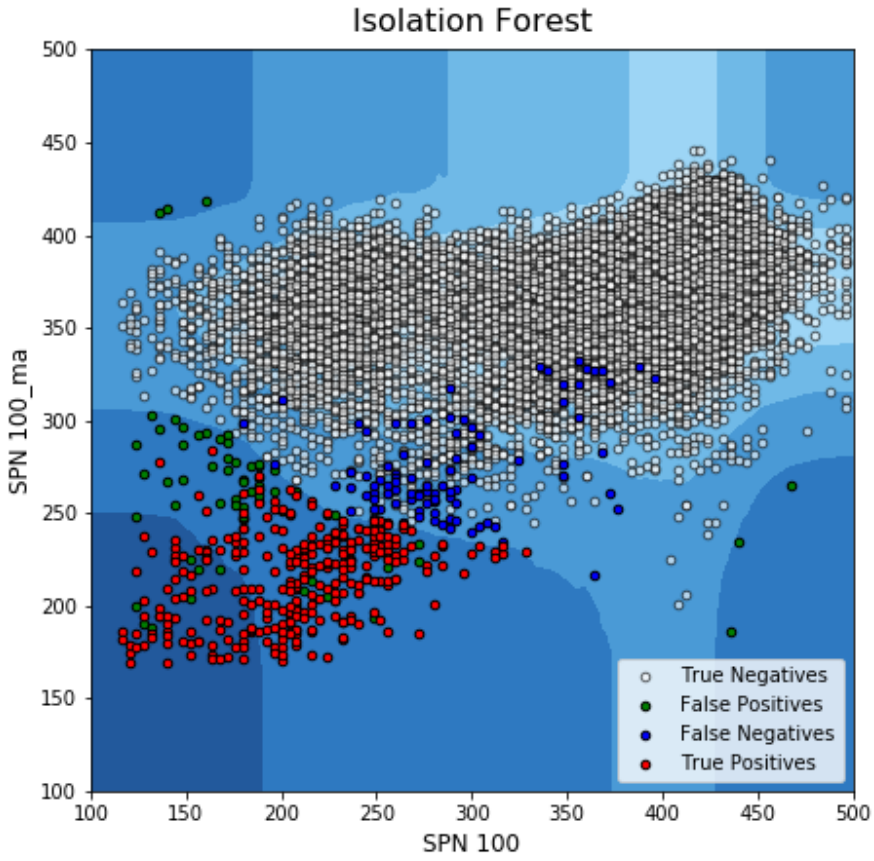


Figure 6.8: Scatter plot showing Isolation Forest predictions. The background indicates the scores of a potential point in that area: the darker the color, the higher the anomaly score. In this case the dataset is divided into training and test (stratified) with rate 0.25.
The statistics are: - *Precision*: 0.767, - *Recall*: 0.765.

SPN 100 has slightly higher values (up to 300) but the moving average hold its values between 250 and 300. Basically, we should weight a little more the moving average feature instead of setting the same weight to both of them. By the way, it is both a limit of the model and a limit of data which are full of non-cleanable noise.

Finally, by applying stratified cross-validation new results happen, even if they are not so different from those in the previous figure. Having a brief look at the figure 6.9, we see that one fold of cross-validation has results around the previous ones.

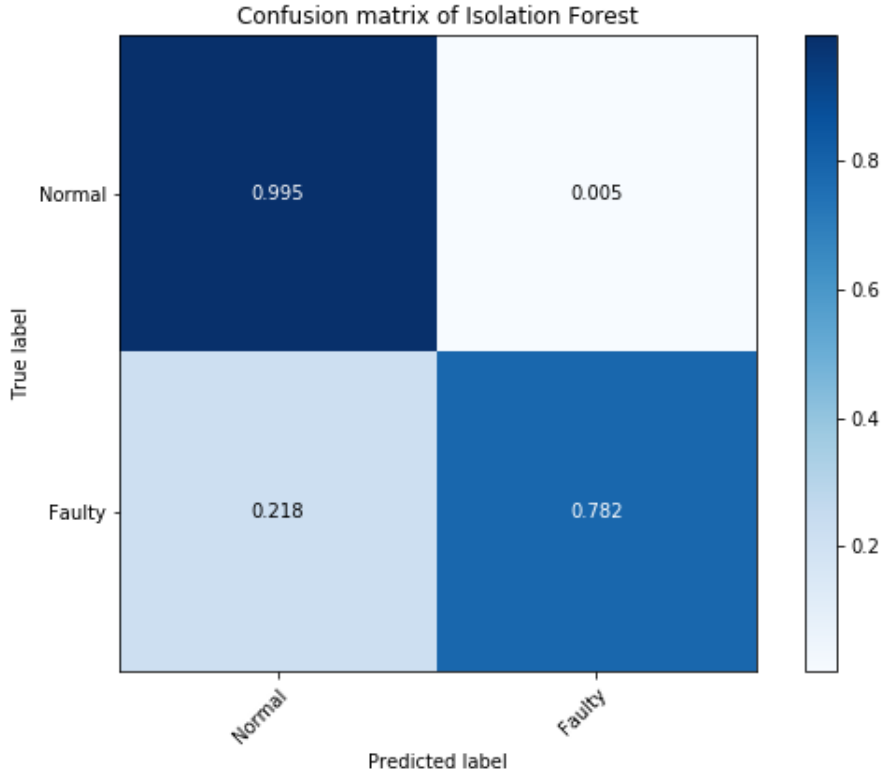


Figure 6.9: Confusion matrix showing Isolation Forest predictions.
The statistics are: - *Precision*: 0.760, - *Recall*: 0.782.

After cross-validation technique, by averaging the statistical scores, we obtain the final result about a general accuracy of the model which provides:

$$Precision : 0.782, \quad Recall : 0.782, \quad AUC : 0.889.$$

6.5 Autoencoder Anomaly Detection using Reconstruction Error

Unlike the previous methods, here we are going to talk about spectral anomaly detection techniques. The approach, generally, is supported by the idea that in a lower dimensional space the two classes are expected to be well separated from each other. So, these kind of algorithms reduces the spaces into a lower one and then each data point is reconstructed, that means a point is brought back into its original form. Then, by the reconstruction error, which is the measure of the distance between the original data and the reconstructed one, each point can be classified as normal or anomalous. The meaning of the procedure is based on the the expectation to obtain the true nature of a data by destroying it and then reconstructing. In fact, the eventual surrounding noise should be eliminated.

Many algorithms might be used for the problem, all the ones which can perform a dimensionality reduction of data. Our choice is to apply a deep autoencoder, as it is done in the article [Che+17]. Autoencoders are artificial neural networks formed by two different parts: encoders and decoders. The first one has the role to decrease the dimension by merging the features, while the second one has to reconstruct data.

So, let's consider a dataset $X = \{x_1, x_2, \ldots, x_n\}$ where each point is $k-$ dimensional $x^i = (x_1^i, x_2^i, \ldots, x_k^i)$. In our case each dimension corresponds to a different SPN, so that in practice we will have $k = 2$. The autoencoder model can be trained to reconstruct its input. So we call reconstruction error the difference between the original data and the reconstructed ones. In order to measure this difference, we will use the common mean square error.
The two parts of an autoencoder are represented by two nonlinear functions f_θ and g_θ, where the first one is referred to the encoder and the other one to the decoder, and θ is the parameter. So, f_θ maps each data point x^i into a lower dimensional latent space Z by compressing x^i into z^i. The compressed latent representation is then brought back by the decoder function g_θ into a reconstructed point $\hat{x}^i$. More

formally,

$$z = f_\theta^l(x) = \sigma_0^l \left(W_0^l (f_\theta^{l-1}(x)) + b^l \right) \tag{6.10}$$

and

$$\hat{x} = g_\theta^l(z) = \sigma_1^l \left(W_1^l (g_\theta^{l-1}(z)) + d^l \right), \tag{6.11}$$

where l is the number of layers (we are assuming that data passes more than one step), $\theta = \{W_0, b, W_1, d\}$ is the set of model parameters, σ_0 and σ_1 are nonlinear activation functions, $W_0 \in \mathbb{R}^{d_{x_{l-1}} \times d_{x_l}}$, $W_1 \in \mathbb{R}^{d_{x_l} \times d_{x_{l-1}}}$, $b \in \mathbb{R}^{d_{x_l}}$ and $d \in \mathbb{R}^{d_{x_{l-1}}}$ are offset bias vectors. We denoted with d_{x_l} the data dimension selected with l layers, meaning that the encoder function at the step with l layers reduces the dimension of a point from d_{x_l} to $d_{x_{l-1}}$. So, the encoder function will do the same but with the inverse order. The basic assumption is that we will use as many steps for the encoder part as for the decoder one.
In order to achieve the best approximations as possible, the autoencoder is trained on a training set to find out which parameters $\hat{\theta}$ minimize the dissimilarity between the original point x and its reconstruction $\hat{x} = g_\theta(f_\theta(x))$. The optimization problem to solve is, consequently, the following

$$\hat{\theta} = \arg\min_\theta \|X - g_\theta(f_\theta(X))\|. \tag{6.12}$$

Since we fixed the *mse* metric, the norm above is the euclidean. So, as part of the network training the loss functions $\mathcal{L}$, defined by the mean square error metric, that has to be minimized is

$$\mathcal{L}_\theta(X, \hat{X}) = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} \left(x_k^i - \hat{x}_k^i \right)^2. \tag{6.13}$$

It is important to claim that inside the encoder the dimension of data has to be reduced from d_{x_l} to $d_{x_{l-1}}$. Otherwise, the function is trivially the identity function, since by using the identity the error between the original point and the reconstructed one is 0. But through the dimensionality reduction technique, the identity function is not suitable for the problem and so we obtain not trivial results.

In the related work where we have to apply the autoencoder model the feature

dimension is 2, so that just one level of layers we could put in. As a result, the reduction goes from 2 to 1 and the problem becomes a little easier to be formulated. In fact,

$$z = f_\theta(x) = \sigma_0(W_0 \cdot x + b) \tag{6.14}$$

and

$$\hat{x} = g_\theta(z) = \sigma_1(W_1 \cdot z + d), \tag{6.15}$$

where $W_0 \in \mathbb{R}^{2\times1}$, $W_1 \in \mathbb{R}^{1\times2}$, $b \in \mathbb{R}$ and $d \in \mathbb{R}^2$. Finally, the activation functions we choose are both the hyperbolic tangent functions (tanh).

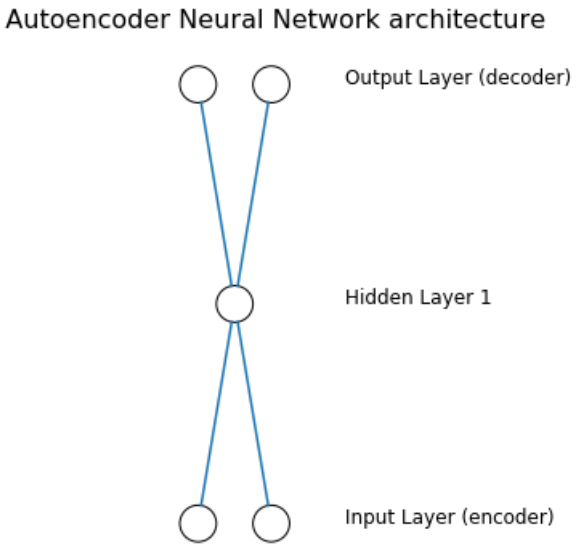


Figure 6.10: Architecture of the autoencoder deep neural network. The input layer and the first hidden layer form the encoder part, whereas the second hidden layer and the output layer represent the decoder. It is a very simple neural network architecture.

The architecture of the neural network is shown in Figure 6.10. It is easy to note that from the point of view of architecture, it is a very simple deep neural network, since there are few layers. What is very peculiar to be a neural network is its use. As we explained how the autoencoder model works, currently we have to pass to its application to the problem of trend anomaly detection. The main idea is that we want to separate the two classes faulty and normal based on the

anomaly score of each data point. So, we would like to have anomalous points with high score and normal data with low score. In order to have it, the fundamental concept is that autoencoder should learn only how to create a normal point. In fact, by training the network on just normal data, we make sure that the decoder part would reconstruct each point as if it was normal because it knows just how to do it. So, basically a faulty data will have a higher reconstruction error because of it will be probably reconstruct as a normal data, with different CAN values. After that, the model is tested on the entire test set and we compute the statistical scores based on a stratified sample of data.

One question has to be solved: how can we compute the threshold separating the two classes into anomalous and normal data? Well, as the previous models, once the anomaly score has been calculated for each CAN message, we have to set the right threshold in order to maximize certain performances of the model. Here it comes the role of the validation set: the idea is to use few data (around 10/15% of the whole data set) to test the trained model in order to find out the optimal threshold which maximizes precision and recall. For this part it is essential not to use the test set, avoiding eventually overfitting of data. Obviously, because of we want to maximize two statistics, the model threshold has to maximize the F_1 score which is the combination of the two previous scores.

Let's summarize how to use an Autoencoder for rare-event classification, as to separate faulty CANs from normal CANs:

- we divide the data into two parts: Faulty CANs and Normal CANs;
- the normal labeled data are treated as normal state of the process. A normal state is when the process is not affected by DM1s;
- we ignore the faulty points of training set and train the autoencoder model on only normal data;
- this Autoencoder has now learned the features of the normal process.
- Then, a well-trained Autoencoder will predict any new data that is coming from the normal state of the process (as it will have the same pattern or

distribution);

- therefore, the reconstruction error of normal points will be small;
- however, if we try to reconstruct a faulty point, the Autoencoder will struggle since it knows only how to recreate normal points;
- this will make the reconstruction error of faulty CANs high;
- we can catch such high reconstruction errors and label them as a faulty CAN prediction.

Now we can go on showing the results. First of all, since the problem has only two features and not so many data, the number of epoch of the training set could be lower than the usual value. In fact, for each epoch the model learn through the training set and evaluate itself on the validation set. But because of the simple structure of data, the loss error tends to be stable just after few epochs, as it is shown in Figure 6.11

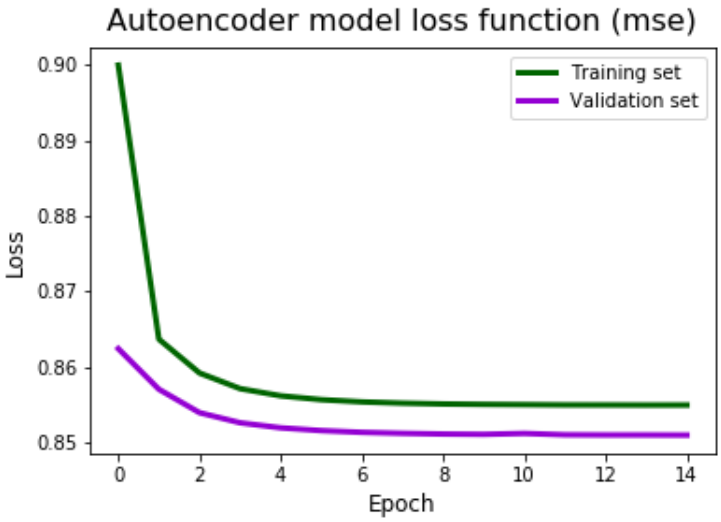


Figure 6.11: Autoencoder loss function calculated on training and validation set. It measures for each epoch the mean square error between the input and the output of the model.

Then after tracing the history of the model, let's have a look at the precision recall curve, shown in Figure 6.12. The optimal threshold is set to 6.27. In order

to find that value, we computed for each threshold the F_1 score. It is an important method to figure out which are the best combination of precision and recall, mostly when we have a lot of couples to examine and many of them are not easy to be compared each other. In fact, generally we should compare couples where, for example, precision of the first is higher than precision of the second but for the recall it's true the opposite relation. So, if the values were the same but inverted, such as $(precision_1, recall_1) = (a, b)$ and $(precision_2, recall_2) = (b, a)$, we would prefer a higher recall. We think it might be more useful to predict faulty when the CAN is normal instead of the contrary. As a result, the recall score is much more important for the problem than the precision.

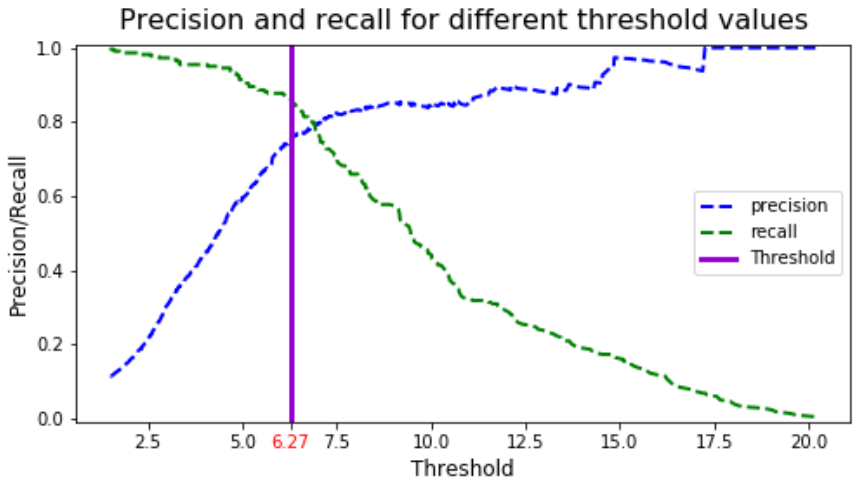


Figure 6.12: Autoencoder precision/recall curve. It shows for each threshold the precision and recall values that the model performs on the validation set. The common method provides to choose the optimal threshold by maximizing the F_1 score on the validation set.

Then, the next step involves the prediction on the test set, by computing for each point its reconstruction error and then labeling it as faulty if the score is higher or equal than 6.27, normal if it is lower. In order to compute the model performances on the test set, look at the Figure 6.13. Green points represent all the true normal data, whereas the faulty CANs are colored red. The violet line is the optimal threshold, which has the equation $(y = 6.27)$. So, the scatter plot shows that each point under the violet line is predicted as normal and that the

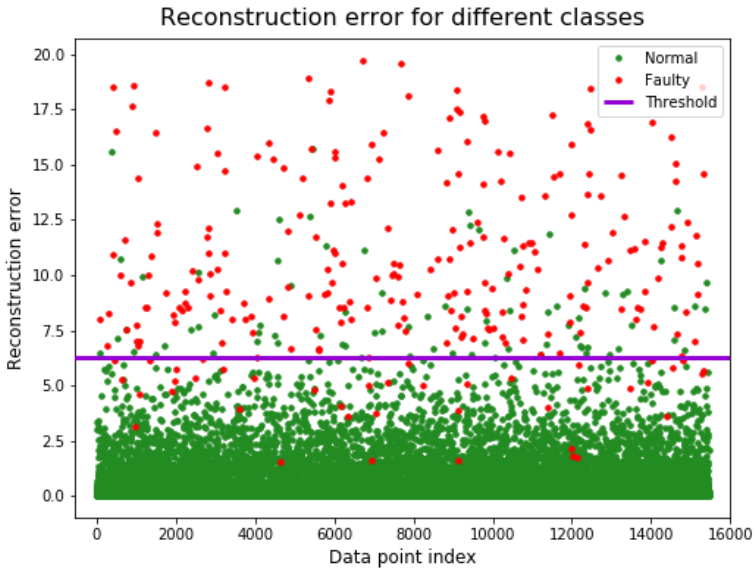


Figure 6.13: Scatter plot showing for each point its reconstruction error. Data points are colored by classes. The threshold divides the predictions of the two classes based on the error value.

model doesn't make so many errors: just few red points are below the threshold, so that we could expect a high AUC score (see Figure 6.5 for the result). In addition, a little more green points are located in the above region, even if the majority of data are red. Overall, the results achieved are pretty good, meaning that the algorithm is well working even if the amount of data is not so elevate and the number of features is 2. By the Figure 6.14, we have a clearer situation of the performance of Autoencoder neural network. Observe that the model predict always faulty when values are inside the faulty gap and rarely it makes a mistake claiming faulty when a DM1 is not close to occur. So, as a result, by using stratified cross-validation on the data set the model reaches the following scores:

$$Precision : 0.802, \quad Recall : 0.796, \quad AUC : 0.994.$$

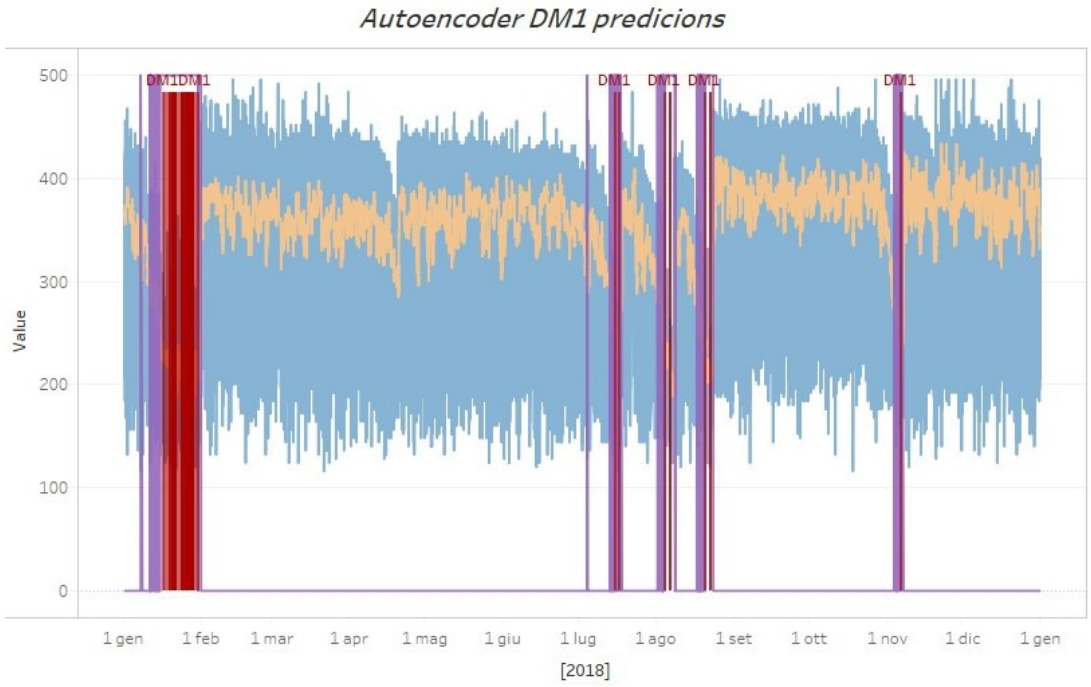


Figure 6.14: Plot showing SPN 100 time series for one year with its moving average (orange). The red lines underscore DM1 occurrences whereas the violet ones highlight Autoencoder predictions (about 30 hours before the first DM1).

6.6 Increasing the interpretability of Autoencoders using Hidden Markov Models

In this section we focus again on Autoencoders, state of the art models in Machine Learning problem solving. Following the recent progress in deep learning, researchers of machine learning are developing many deep models based on Autoencoders. Furthermore, the importance of understanding and interpreting what goes on deep neural networks gives value to the search for new methods to make the models more complete. Our approach to increasing interpretability is by combining an Autoencoder with a Hidden Markov Model (HMM), a simpler and more transparent model. So, what could be the role of HMM for the new model?

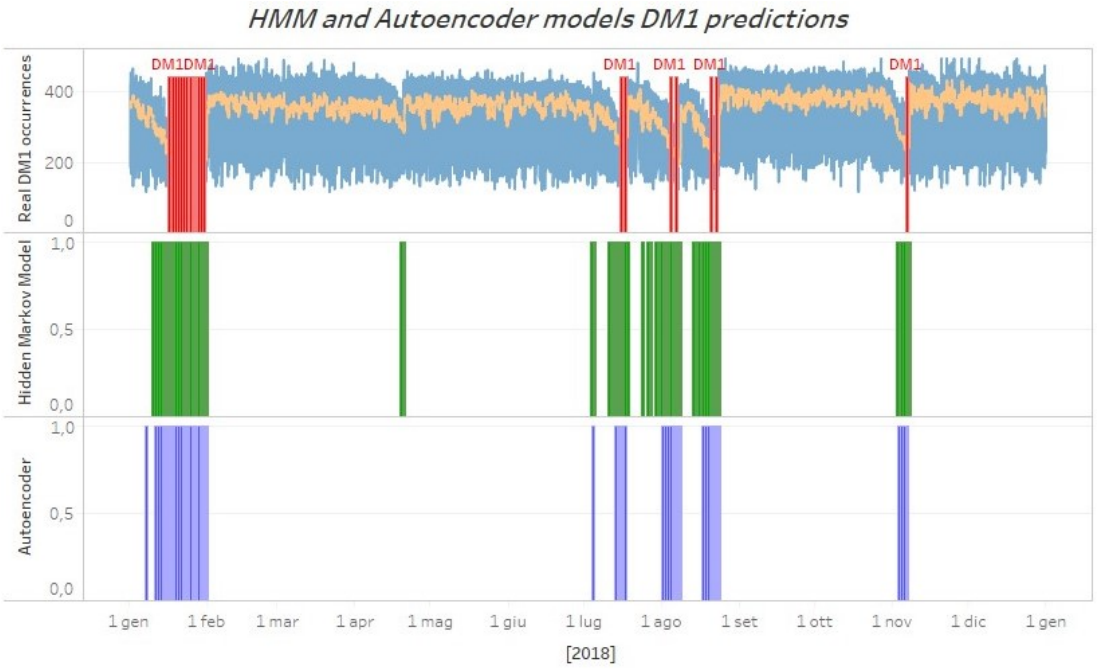


Figure 6.15: The first graph shows an example of SPN 100 trend together with its moving average (orange). The red lines identify when a DM1 occurs. The second plot illustrates the Hidden Markov Model predictions about anomalous trend, while the violet plot is related to Autoencoder predictions. The sample involves data from mid January 2018 to January 2019 and just one unit is considered.

Currently we have a dataset where the response variable of the models is binary

and its values are chosen by an empirical and deterministic approach. We selected the best time when CANs start changing substantially their values and we put that time as a separation of the two classes faulty and normal. But, can we automate this mechanism by searching for a method which is independent of the time but looks just for the significant variation in the CAN trends? Well, as we explained in a chapter before, Hidden Markov Models can be trained exactly for this aim. Looking at the Figure 6.15, we can observe the different roles that HMM and Autoencoder have. In fact, while HMM is less accurate but it is able to predict DM1 until 5 days before, Autoencoder is very precise but it predicts DM1 just 30 hours in advance and it has to be trained on a well defined normal behaviour of data. By the way, the new two-steps model has the following form:

- First, we train a HMM on the training set and through its prediction of the states we label a data point as faulty or normal, like in section 5.5.2.
- Then we assume that the prediction is true, in the sense that the trend is changing state for real.
- As a result, we train the Autoencoder model on the training set, using the labels defined by the HMM.
- Finally, in order to evaluate the performances we looks for the true empirical response variable defined with time label $30.0h$.

Actually, the sense of this approach regards the interpretability of data. Since the goal is to predict within more or less a range of 30 hours the occurrence of a DM1, the data into the training set are affected by a lot of noise. It causes many mistakes due to the fact that most of the CANs close to the threshold between the classes (we mean those messages which are broadcast about 30 hours before) are assigned to a class whereas it is likely that they belong to the other one. So, instead of using a deterministic classification, we take the HMM results as they were true to train the deep neural network model.
Finally, the results computed by using cross-validation are shown below:

$$Precision : 0.810, \quad Recall : 0.808, \quad AUC : 0.995.$$

By comparing the results with those obtained by Autoencoder without assuming the HMM output as true status, we could observe that they are slightly better. In fact, probably the normal situation is much better defined because all the points with an anomalous trend are considered as faulty even if didn't appeared any DM1. So, the training is performed at the top. The boundary is related to the test data points which are hard to classify correctly, according to the noise surrounding them.

Chapter 7

Testing the model on unknown DM1s

The whole model seems to work when one DM1 has data from its related parameter inside the dataset. But, as we said in the section 2.3.2, without the suggestions by a domain expert we probably would choose the 3 DM1s which frequently occur together. It might be a relevant choice because their occurrence at the same time could be a symptom of of a more serious problem. So, in order to keep the same structure as before, we consider those 3 DM1s as a unique more complex DM1 and we are going to repeat the whole procedure to predict it.

We will skip the data preparation part because it is exactly the same as the other DM1, so that we start discussing the results from the feature selection part.

7.1 Feature Selection for unknown DM1s

Following the previous approach, we have to apply the Mutual Information, SVM-RFE and ERT algorithms. So, without repeating again the theory behind all the methods, let's use the same thresholds as for the engine oil pressure DM1. The results are far from the previous ones: since in this case the dependence is really low, the final table representing the features passing all the tests is empty. So, in

	label_time	feature	mutual_info_score	importance_ETC	rank_RFECV
0	1.0h	30036	0.31	10.43	1.0
1	2.0h	30036	0.30	10.64	1.0
2	2.0h	30104	0.21	10.00	1.0
3	2.5h	30036	0.32	10.40	1.0
4	3.5h	30036	0.30	10.55	1.0
5	5.0h	30036	0.31	10.43	1.0
6	5.0h	30104	0.22	10.68	1.0
7	12.0h	30104	0.23	10.59	1.0
8	30.0h	30036	0.30	10.48	1.0
9	30.0h	30104	0.30	9.94	1.0
10	36.0h	30036	0.31	10.48	1.0
11	36.0h	30104	0.26	9.60	1.0

Table 7.1: Table showing the dataframe containing only those relevant features and the respective time label.

order to have at least one feature, we decide to reduce thresholds to:

$$\text{Threshold Mutual Information} = 0.2\,\%; \tag{7.1}$$

$$\text{Threshold Importance ERT} = 10.0\,\%; \tag{7.2}$$

$$\text{Threshold ranking RFE-SVC} = 1.0. \tag{7.3}$$

Proceeding in this way, the results are shown in the table 7.1. See that just two SPNs are relevant for the problem, even if they obviously are low dependent from the response variable (low scores). The CAN parameters number 30036 is related to the *Engine Battery Voltage*, whereas the number 30104 measures the *Engine Urea Tank Level*. Since they both are not qualitative variables, we add two columns containing their moving averages. Before going on, note that choosing the RFE-SVC threshold equal to 1 is not restrictive. In fact, as it is shown in Figure 7.1, through the cross-validation score exactly 12 features are selected and ranked as first. The main reason is always the same, that is the weak dependence.

Let's move on the next step, which corresponds to the time series distance by DTW. The technique considers one SPN at the time among 30036, 30036_*ma*, 30104 and 30104_*ma*. Then, it computes the DTW score by the rate of faulty

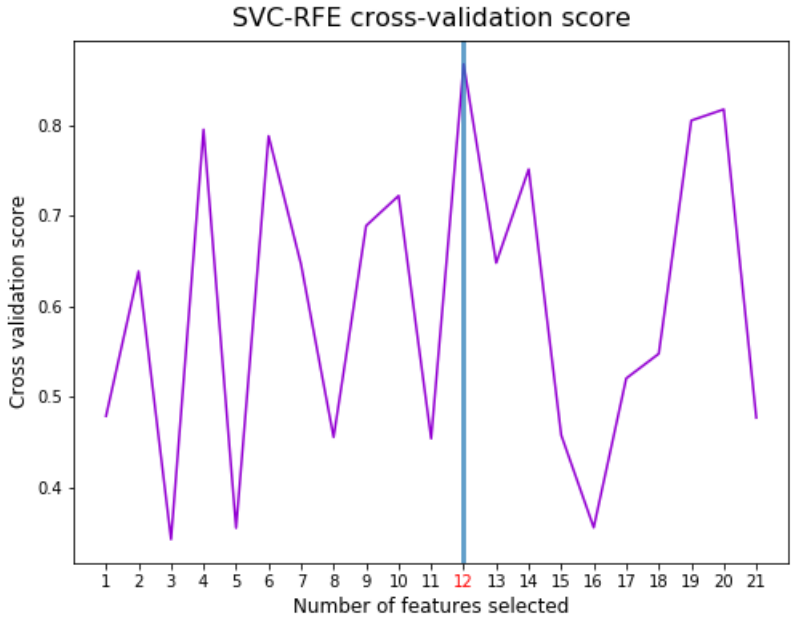
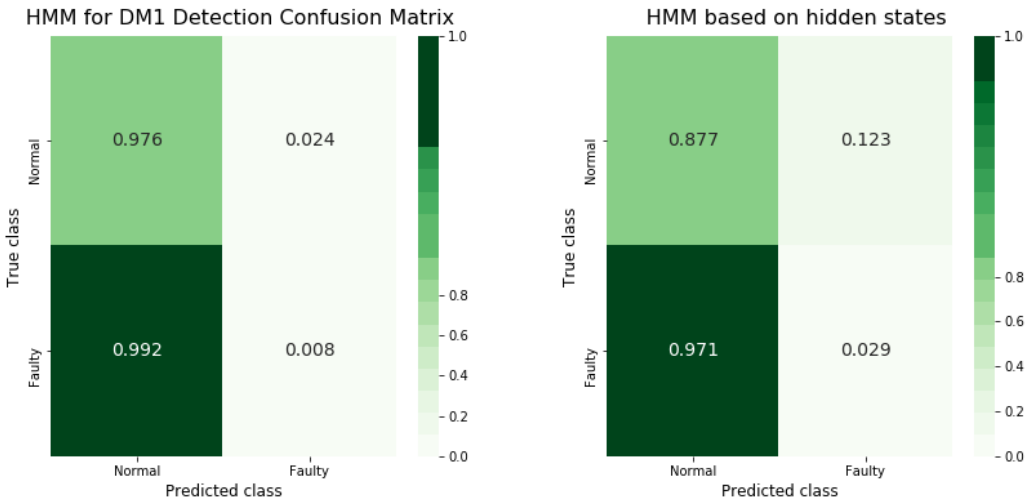


Figure 7.1: Plot showing for each number of selected features its cross-validation score. The best ranking is chosen according to the maximum score.

windows over the total number of time series (see the section 4.6.2). If the same threshold as the previous case is used, then the result is an empty set of features. So, let's use the threshold of 0.25. The results shown in table 7.2. As you can see there, just SPN 30036 is a little significant from the point of view of sequential values. It means that neither SPN 30104 nor the moving averages are very relevant for the problem. But in order to use the methods we developed, we are going to add to the SPN 30036 its moving average, so that the dimensions will be at least 2.

	spn	time_label	faulty_can	count	rate Faulty/Normal
1	30036	1.5h	10.0	39	0.256410
2	30036	2.5h	10.0	39	0.256410
3	30036	3.0h	11.0	38	0.289474
4	30036	4.5h	10.0	37	0.270270
5	30036	5.0h	10.0	37	0.270270

Table 7.2: Table showing the dataframe containing only those relevant features and the respective time label for the DTW score.



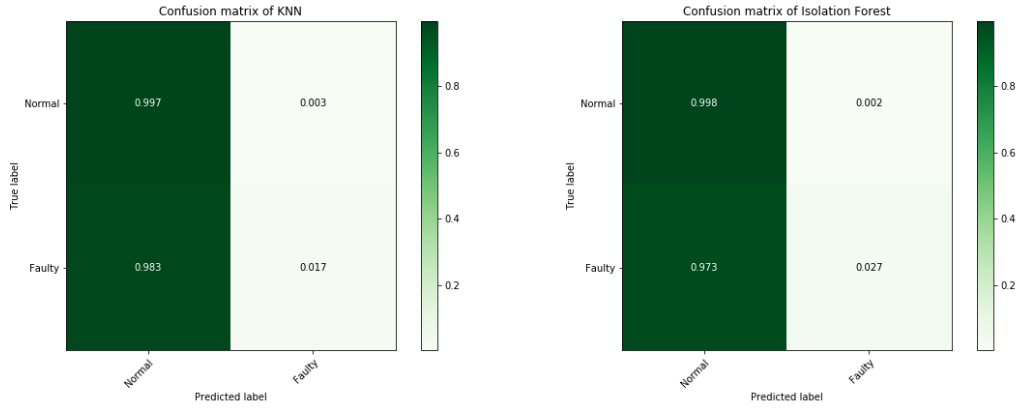
(a) (b)

Figure 7.2: (a) At the left, confusion matrix showing the scores related to HMM probabilities to detect an unknown DM1. The statistics are: - *Precision*: 0.0001, - *Recall*: 0.0083. (b) At the right, confusion matrix showing the scores related to HMM states predictions over an unknown DM1. The statistics are: - *Precision*: 0.0006, - *Recall*: 0.0289.

7.2 Predictive models results

Starting from the Hidden Markov Model, we hope that all the faulty CANs belong to the same state. Looking at the confusion matrix in the Figure 7.2b, you can immediately imagine that the expected situation doesn't occur: since the true positive rate is really low, faulty CAN messages are distributed over many different states so that the HMM can't be used as a classifier. Both the two confusion matrices are reported in Figure 7.2, showing the extremely low values achieved with this kind of data. As a result, the model isn't efficient to predict and prevent a failure.

Moving on to the kNN Outlier Detector model, let's find if the distance-based model might be more effective. So, the number of neighbours is put to the amount of faulty CAN messages plus 100. The contamination score, as in the Isolation Forest, is the ratio of the number of faulty CANs to the total number of CANs



(a) (b)

Figure 7.3: (a) At the left, confusion matrix showing the scores related to kNN Outlier Detector (distance based algorithm). The statistics are: - *Precision*: 0.0147, - *Recall*: 0.0143. (b) At the right, confusion matrix showing the scores related to Isolation Forest, which is the tree based model. The statistics are: - *Precision*: 0.0247, - *Recall*: 0.0247.

inside the training set. In addition, we test the model also with the Isolation Forest technique, which is connected to the tree based approaches. Both the results are reported in the Figure 7.3.

The last method we test is the Autoencoder model. Since we have left two features inside the dataset to use properly this neural network, let's set the hidden dimension to 1 and use one encoder and one decoder. The number of epochs is set to 25 and the procedure is cross-validated. As a result, look at the Figure 7.4 to figure out the scores.

As the previous algorithms, low scores have the meaning of low dependence between the features and the response variable. Basically, from this chapter it emerges that if the chosen DM1 has the related SPN parameter monitored, then the model might achieve high results and it might be efficient for the real prediction. On the other hand, if the SPN is not controlled then the analysis shall not be made as in this thesis.

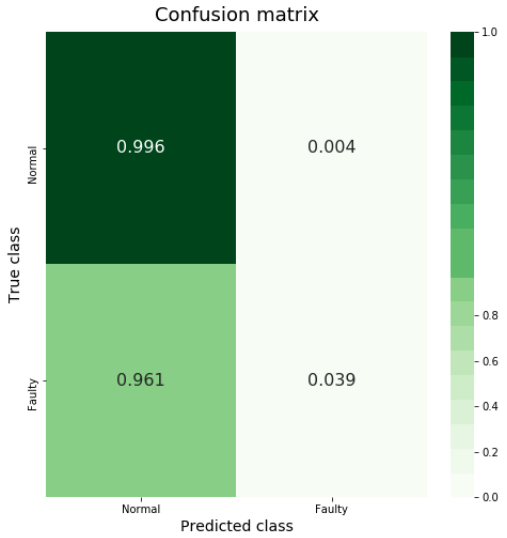


Figure 7.4: Confusion matrix showing the Autoencoder final statistics: - *Precision*: 0.0252, - *Recall*: 0.0366, - *AUC*: 0.595.

Chapter 8

Final evaluation

8.1 Overall results

After data preparation, which refers to the construction of an appropriate dataset for the prediction, the feature selection process outputs are really interesting. As we discussed above, just one SPN (100) is dependent from the engine oil pressure DM1 even if its measure of dependence is very low, meaning that a lot of noise affects data.
Now we can compare all the results obtained with the algorithms above. The final scheme is reported in Figure 8.1.

Looking at the tree map, we see that the Markov model based on the emission probabilities is not competitive with respect to the others. Then, taking hidden states into account, the precision, recall and AUC scores dramatically increase.
Distance based method called largest kNN achieves scores around 70% (precision and recall), highlighting its limits related to the use of distance as anomaly score. In fact, our problem has to detect just low values and not extreme points in a wider sense, as the kNN model might do.
The isolation forest algorithm divides the space into a sort of grid by cutting each feature once at the time. Then, the outliers are those points which are located in a less density zone. So, the results are quite high because of the method is able to separate low values from the normals through sequential cuts. The limit of the

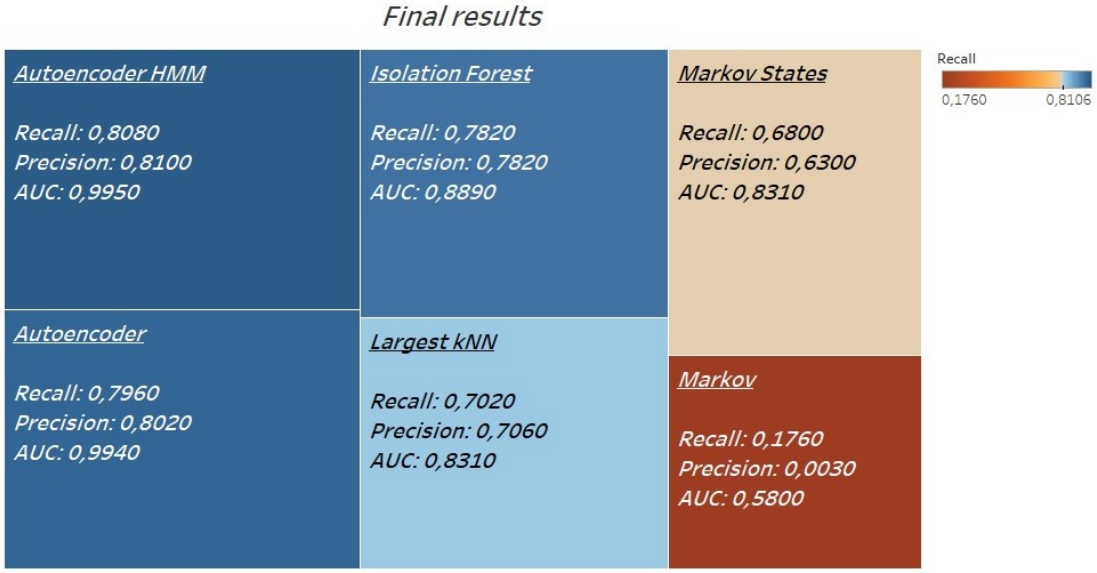


Figure 8.1: Tree map showing for each model its scores (Precision, Recall, AUC). Colors are based on recall score so that the blues are higher than browns as reported in the legend. About the dimensions of the rectangle it has to be read as the bigger the higher AUC score.

model is to give the same weight to the variables, so that when just one of the two SPNs is low it predicts faulty.

With the Autoencoder model the scores are almost to the top. Basically, it's limits concerns the amount of data which could not be sufficient to perform perfect results. Furthermore, its performances might decrease because of the 2 dimensional space. In this case, even the presence of noise surrounding all the points could affect the results.

By training Autoencoders on Hidden Markov Model output, the interpretability of the problem increase because what we did is to consider variable time interval before the DM1 instead of putting a deterministic and empirical one. So, it means that the situation is completely disjointed from the time context, even if we want to make a prediction with a gap of 30 hours.

Finally, the two step model is the best in term of performances, but it requires a lot of data and features to perform at the top.

8.2 Conclusion

Off-road vehicles usually work in suburbs where mechanics can't get there in a short time. So, unplanned stops could make them loose a large amount of time. For this reason a properly maintained vehicle might reduce the risk of errors and keep the unit working without wasting time. Predictive maintenance is the answer to those problems because it is able to introduce in-time recommendations which might limit the waste of time and make works more efficient. Here we have presented a mathematical approach based on HMMs and Autoencoders for predicting upcoming error messages about the engine oil pressure too low. The model is formed by using the currently available vehicle CAN messages together with the related DM1s. Consider that this data are taken day by day and they are designed for other purposes. Actually, they are not used for predictive maintenance but just for statistical analyses. It creates many challenges, as the one connected to the granularity of data and the connection between the parameters messages and the error messages we dealt with in this work. This challenge is presented and handled in order to build a right predictive model.

As we said before, the research contribution can be divided into two parts: granularity of data and dependence between SPNs and DM1. About the first one, since what we would want to do is to build a predictive model, we should have as more precise as possible data and, mostly, as much data as we can . In order to make a really accurate data cleaning, messages should be taken every one or two minutes at the most. Currently the granularity is 10 minutes, so that we don't have many data and they are full of noise since each point is a statistic on a 10-minute-sample (about 60.000 records). In fact if we selected data into an accurate interval of normal CANs (no faulty) which probably doesn't contain any outliers or dirty points, the result would be as in picture 8.2. It illustrates how values would be after we cleaned as more precise as we can all the normal points, and the difference between the amount of data moving from 10 minutes to 2 minutes of granularity. From this we conclude that, while we can't do it currently because of the small number of 10 minutes data, by changing the granularity to 2 minutes we might have enough points to make the same analysis as before but with a cleaner and

more accurate dataset. The last observation about the amount of data is connected

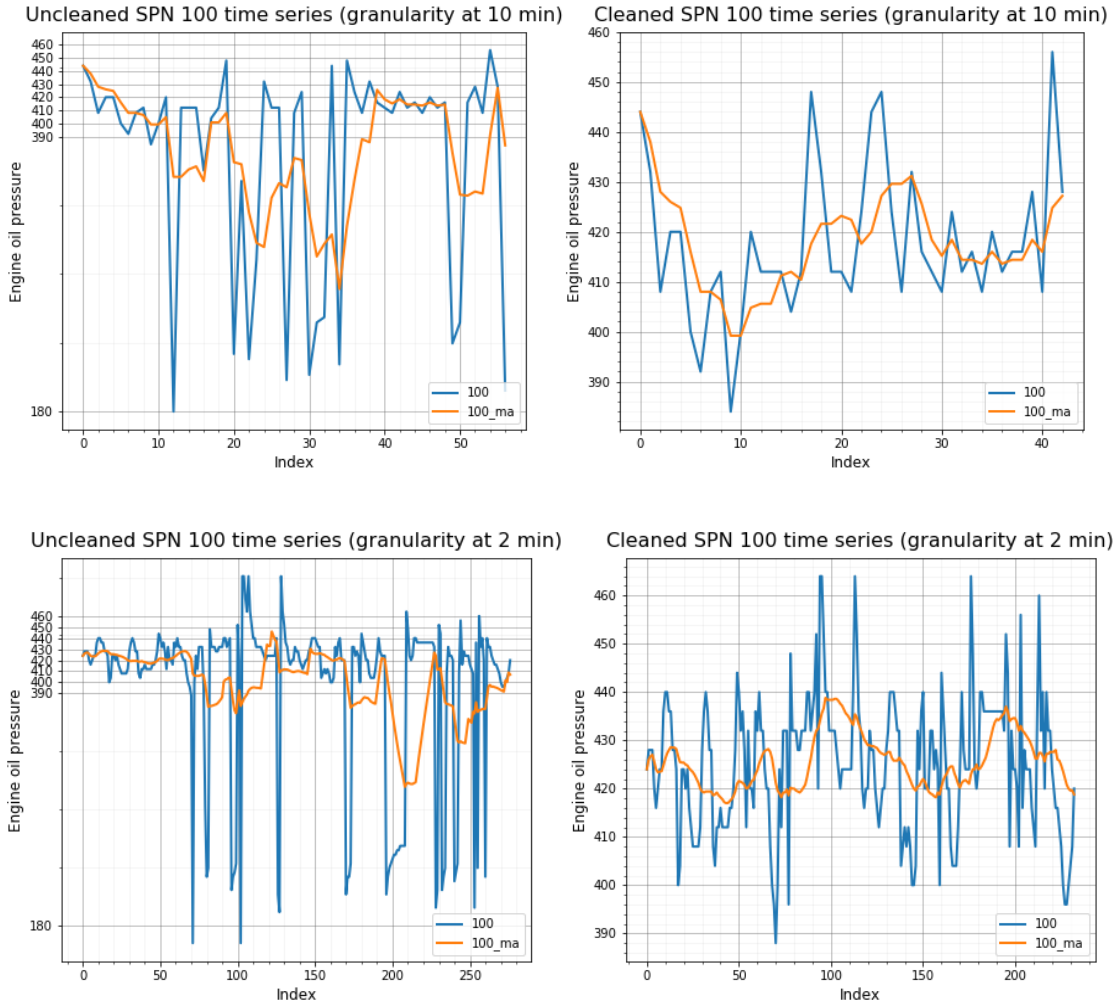


Figure 8.2: The graph shows two different behaviours under granularity at 10 minutes (above) and 2 minutes (below). Data are taken during 30 hours randomly selected. In the left part, you can see the time series plot without accurately cleaning data, whereas at the right it is reported the plot after filtering points just to values into the interval $[375, 475]$, which is the most likely to contain only real explainable data. Looking from the left to the right, you can observe how the time series becomes significantly more precise. Looking from top to bottom, it is evident the increase of the amount of data (from 57 to 277 and 43 to 233).

with the procedure we assume inside this thesis. Since to mark each CAN message

with Normal or Faulty labels the method goes backward in time until 30 hours are reached, we feel it has to be remarked that the number of messages influence a lot the entire model, mostly the feature selection part. Suppose that one SPN predicts the DM1 about 1 hour in advance or even 30 minutes. Then, just 3 or 4 or 5 messages are likely to be labelled as faulty. It means that we should compute distributions or sequentiality measures with just few data and it might negatively affect the results. For this reason, a lower granularity is needed.

The second contribution is a practical demonstration of the uncorrelation, in the wider sense of non-dependence, between SPNs and DM1s as it emerged from the chapter 7. Generally the applied feature selection methods eliminate all the SPNs but the one which directly influences the DM1. In fact, one experiment involving one unknown DM1 revealed that no SPNs are connected with the response variable. Of course it means that, through the same thresholds we put, the result was an empty dataframe. So the suggestion we can propose is to change the kind of recorded parameters. Since the dependence is very low even if the SPN is directly involved into the DM1 occurrence, in order to have an accurate prediction they should take care of as many as possible parameters related to the DM1 of interest. So, for first it is necessary to detect a relevant DM1 through the knowledge of the domain, and then they could put sensors that detect DM1-related data to get more information about the problem. It will increase substantially the accuracy of the model, so that a valid implementation could be performed.

Bibliography

[Rab89] Lawrence R Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286.

[CT91] Thomas M Cover and Joy A Thomas. "Entropy, relative entropy and mutual information". In: *Elements of information theory* 2 (1991), pp. 1–55.

[YXC94] Jie Yang, Yangsheng Xu, and Chiou S Chen. "Hidden markov model approach to skill learning and its application to telerobotics". In: *IEEE transactions on robotics and automation* 10.5 (1994), pp. 621–631.

[DD99] Richard Durrett and R Durrett. *Essentials of stochastic processes*. Vol. 1. Springer, 1999.

[RRS00] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. "Efficient algorithms for mining outliers from large data sets". In: *ACM Sigmod Record*. Vol. 29. 2. ACM. 2000, pp. 427–438.

[ZM00] Xinchuan Zeng and Tony R Martinez. "Distribution-balanced stratified cross-validation for accuracy estimation". In: *Journal of Experimental & Theoretical Artificial Intelligence* 12.1 (2000), pp. 1–12.

[KP01] Eamonn J Keogh and Michael J Pazzani. "Derivative dynamic time warping". In: *Proceedings of the 2001 SIAM international conference on data mining*. SIAM. 2001, pp. 1–11.

[BDC02] Peter J Brockwell, Richard A Davis, and Matthew V Calder. *Introduction to time series and forecasting*. Vol. 2. Springer, 2002.

[Cha+02] Nitesh V Chawla et al. “SMOTE: synthetic minority over-sampling technique”. In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.

[Guy+02] Isabelle Guyon et al. “Gene selection for cancer classification using support vector machines”. In: *Machine learning* 46.1-3 (2002), pp. 389–422.

[HPL02] Steve Corrigan HPL. “Introduction to the controller area network (CAN)”. In: *Application Report SLOA101* (2002), pp. 1–17.

[Sta04] Mark Stamp. “A revealing introduction to hidden Markov models”. In: *Department of Computer Science San Jose State University* (2004), pp. 26–56.

[Sal05] Felix Salfner. “Predicting failures with hidden Markov models”. In: *Proceedings of 5th European Dependable Computing Conference (EDCC-5)*. 2005, pp. 41–46.

[GEW06] Pierre Geurts, Damien Ernst, and Louis Wehenkel. “Extremely randomized trees”. In: *Machine learning* 63.1 (2006), pp. 3–42.

[SC07] Stan Salvador and Philip Chan. “Toward accurate dynamic time warping in linear time and space”. In: *Intelligent Data Analysis* 11.5 (2007), pp. 561–580.

[Fur08] Titus Felix Furtună. “Dynamic programming algorithms in speech recognition”. In: *Revista Informatica Economică nr* 2.46 (2008), p. 94.

[LTZ08] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. “Isolation forest”. In: *2008 Eighth IEEE International Conference on Data Mining*. IEEE. 2008, pp. 413–422.

[CL11] Chih-Chung Chang and Chih-Jen Lin. “LIBSVM: A library for support vector machines”. In: *ACM transactions on intelligent systems and technology (TIST)* 2.3 (2011), p. 27.

[LTZ12] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation-based anomaly detection". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 6.1 (2012), p. 3.

[Rak+12] Thanawin Rakthanmanon et al. "Searching and mining trillions of time series subsequences under dynamic time warping". In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2012, pp. 262–270.

[Ota+13] Dr Otair et al. "Approximate k-nearest neighbour based spatial clustering using kd tree". In: *arXiv preprint arXiv:1303.1951* (2013).

[Pry14] Rune Prytz. "Machine learning methods for vehicle predictive maintenance using off-board and on-board data". PhD thesis. Halmstad University Press, 2014.

[SZZ15] Skyler Seto, Wenyu Zhang, and Yichen Zhou. "Multivariate time series classification using dynamic time warping template selection for human activity recognition". In: *2015 IEEE Symposium Series on Computational Intelligence*. IEEE. 2015, pp. 1399–1406.

[YZ15] Ke Yan and David Zhang. "Feature selection and analysis on correlated gas sensor data with recursive feature elimination". In: *Sensors and Actuators B: Chemical* 212 (2015), pp. 353–363.

[A+16] Preeti Arora, Shipra Varshney, et al. "Analysis of k-means and k-medoids algorithm for big data". In: *Procedia Computer Science* 78 (2016), pp. 507–512.

[Ye+16] Ning Ye et al. "Vehicle trajectory prediction based on Hidden Markov Model." In: *KSII Transactions on Internet & Information Systems* 10.7 (2016).

[Che+17] Min Chen et al. "Deep features learning for medical image analysis with convolutional autoencoder neural network". In: *IEEE Transactions on Big Data* (2017).

[Mur17] Yonathan Murin. "k-NN Estimation of Directed Information". In: *arXiv preprint arXiv:1711.08516* (2017).

[Fol+18] Duarte Folgado et al. "Time Alignment Measurement for Time Series". In: *Pattern Recognition* 81 (2018), pp. 268–279.

[GE18] Izhak Golan and Ran El-Yaniv. "Deep Anomaly Detection Using Geometric Transformations". In: *Advances in Neural Information Processing Systems*. 2018, pp. 9758–9769.

[PAD18] Stanislav Pidhorskyi, Ranya Almohsen, and Gianfranco Doretto. "Generative Probabilistic Novelty Detection with Adversarial Autoencoders". In: *Advances in Neural Information Processing Systems*. 2018, pp. 6822–6833.

[KD] Marius Kloft and BERLIN DE. "Hidden Markov Anomaly Detection". In: ().