Learning Error Patterns from Diagnosis Trouble Codes*

Stefan Kriebel[2], Evgeny Kusmenko[1], Bernhard Rumpe[1], Igor Shumeiko[1]

Abstract— **Diagnostic trouble codes (DTCs) are steadily produced by a vehicle's control units to support the diagnosis process when the vehicle is maintained or to initiate predictive maintenance. Although, DTCs carry a lot of information, possibly including environmental data such as the engine temperature, the velocity, etc., they are of little help to an automotive engineer if seen without a context. In fact, a concrete problem can mostly be diagnosed if an already known pattern of DTCs is present. However, detecting new patterns in masses of vehicle data gathered each day from thousands of vehicles and recognizing known patterns accurately cannot be performed manually by automotive engineers. We propose an unsupervised DTC pattern learning framework supporting the daily field data analysis of original equipment manufacturers (OEMs).**

I. Introduction

Vehicle maintenance heavily relies on field data a vehicle has gathered using its On-board diagnosis (OBD) equipment. Whenever an abnormal state is detected by the vehicle's electronics, a corresponding Diagnostic Trouble Code (DTC) is saved as a hexadecimal identifier together with the environmental conditions if applicable for later readouts. For instance, the charging electronics of an electrical vehicle measuring an under-voltage while charging would save the DTC related to under-voltage as well as the actually measured voltage value.

Although, DTCs capture information describing the symptoms as precisely as possible, they fail to reveal the underlying reason. Observing under-voltage during the charging process can be caused by a multitude of problems. Experienced automotive engineers rather learn and detect recurrent *DTC patterns*. However, detecting new patterns in masses of vehicle data gathered each day from thousands of vehicles and recognizing known patterns accurately during maintenance cannot be performed manually. Growing fleets sending their OBD data to the manufacturer on a regular basis require automated techniques learning new patterns and recognizing them precisely. **The main contribution of this work is an unsupervised error pattern learning framework supporting the daily field data analysis of original equipment manufacturers and ensuring that pattern knowledge is not bound to a small set of experts, but is made explicit and reusable for further analyses.** The learned patterns can be used not only during maintenance, but also by other vehicles for a better self-diagnosis. This way, an intelligent vehicle can pro-actively recommend a maintenance or, in easily solvable cases, advise the owner how to fix a problem.

*This work was supported by the Grant SPP1835 from DFG, the German Research Foundation.

[1]Department of Software Engineering, Faculty of Computer Science, RWTH Aachen University, Ahornstraße 55, 52074 Aachen, Germany {kusmenko,rumpe,shumeiko}@se-rwth.de

[2]BMW Group, Munich, Germany, stefan.kriebel@bmw.de

II. Preliminaries

During its lifetime, a vehicle produces a whole lot of OBD data which can turn out to be important or irrelevant during diagnosis. The DTCs gathered by a vehicle are either sent to the OEM via a mobile connection or are read in a repair shop using a compatible diagnosis tool. The thereby obtained data set is then saved in a central database for further analysis as a so called *readout*. We define a readout as

$$R = (v, D, date, distance) \tag{1}$$

where v is the vehicle the readout comes from, D is a set of DTC *instances*, $date$ is the date of the readout up to some required resolution and $distance$ is the total distance traveled by v. We furthermore define $\mathcal{D}$ as the set of *all* DTC classes predefined by the manufacturer. Each element $\delta \in \mathcal{D}$ is an abstract representation of a DTC including an identifier as well as a set of environment variables inherent for this DTC. For instance, if $\delta_1 \in \mathcal{D}$ represents the under-voltage DTC, it obviously should have the measured voltage in the set of its environment variables. Furthermore, most DTCs also contain the occurrence date as well as traveled distance at occurrence. We define a function $class : D \rightarrow \mathcal{D}$ mapping a given DTC instance to its DTC class.

Without loss of generality, we can assume for simplicity of notation that a DTC instance is never contained by more than one readout, i.e. $\forall R_1 = (v, E_1, date_1, distance_1), R_2 = (v, E_2, date_2, distance_2) : E_1 \cap E_2 \neq \emptyset \Leftrightarrow R_1 = R_2$. This can easily be ensured technically, e.g. by deleting the saved instances from the vehicle's memory after a readout, or by setting a flag variable for already read DTC instances.

We define a pattern *class* PC as a first-order logic formula where atoms are DTC classes. Using DTC classes as atoms in a logical expression is informal but leads to a very compact and convenient notation. A DTC class atom is evaluated to true in such a formula if an instance of this class is present in a reference set of DTC instances D, e.g. the DTC instances of a readout. We call a set of DTC instances D a pattern *instance* of PC if PC can be evaluated to true with respect to D. For instance, a pattern class $PC_1 = \{\delta_1 \wedge \delta_2 \wedge \neg\delta_3\}$ describes a pattern which is present in a vehicle if it has a DTC instance of δ_1 and of δ_2, but no instance of δ_3, or:

$$\forall d \in D . \exists d_1, d_2 \in D : \tag{2}$$

$$class(d_1) = \delta_1, class(d_2) = \delta_2, class(d) \neq \delta_3). \tag{3}$$

Assume $\mathcal{R}_t$ is a set of readouts, e.g. all readouts of a fleet until a point in time t. Our analysis goal is to learn systematically reoccurring patterns from $\mathcal{R}_t$ which might describe problems better than isolated DTCs.

III. Case Clustering

To learn DTC pattern classes from DTC instance sets we need to agree on which DTC instances belong to a single set or when a DTC instance set needs to be split up in multiple independent sets. For instance, we can consider all DTC instances produced by a car during its lifetime as a self-contained set. In this scenario the number of data points we would use for our analysis equals the number of vehicles in the field. However, we know from experience that DTC instances from different readouts are rarely connected to each other. So, another option is to only consider DTC instances coming from the same readout as a self-contained data point. This not only increases the amount of data, but also ensures that the data points are more concise and less information related to different underlying causes is mixed in a single data point.

However, we may still argue that building data points based on whole readouts might still be too coarse-grained. Usually, we can assume that only DTC instances produced during a short period of time and distance of travel share a common cause. Given a set of DTC instances D we need to be able to subdivide it into partitions so that each of the resulting subsets only contains DTC instances featuring a temporal and/or spatial similarity. Thereby, similarity should be transitive: If a pattern is produced by a chain of events, the chain should not be ripped apart because the first and the last event have a too little similarity. What matters is that the (temporal) distance between neighbor events is small enough.

Therefore, we employ the single linkage clustering technique known for its chaining property [1] to split up the DTC instances of a vehicle into groups of related events. Consider a series of DTC events depicted in fig. 1. A flash denotes an occurrence of a DTC, i.e. a creation of a DTC instance. The corresponding DTC class is given by a hexadecimal identifier above the flash. The small red houses denote readouts at which the previous DTC instances up to the preceding readout are read. In this example we assume that two DTC instances are grouped together only if they lie not more than one week and 50 km apart. However, these two parameters highly depend on the use case and should be chosen carefully based on the engineers' experience: some patterns arise during a timespan of a few seconds while some evolve much slower throughout an extended period of time.

As we can see, the DTCs `0x5` and `0x7` end up in the same cluster denoted by a blue ellipse as they lie only two days apart and occurred within a travel distance of 50 km. The same holds for the cluster containing the DTCs `0x3`, `0xA`, and `0x8`. Note that this second cluster is kept together by the chaining effect of average linkage clustering: being ten days apart, the DTCs `0x3` and `0x8` are not similar as they violate the temporal similarity constraint. However, they are held together by the DTC `0xA`.

Assuming there are no more readouts and DTC instances available for this vehicle this means that we subdivided its data into two separate self-contained entities which can be regarded as data points in our analysis later on. In the following we refer to such an entity carrying the information of one data point as a *case*.

As we have discussed, we have three ways to create cases. First, a case can contain all DTC instances of one car. Second, a case can contain all instances of a readout. Third, the DTC instances are clustered based on a similarity measure, most often the distance traveled and time. Of course, arbitrary further strategies are possible, e.g. taking into account software updates, but are not in the scope of this paper. Furthermore, case creation strategies can be combined. For instance, we assume that both the separation of DTCs based on readouts as well as on the distance is important. Therefore, in our work we achieved the best results by combining these two strategies. In our example the combination leads to three cases since the cluster containing `0x5` and `0x7` is now separated by the readout `RO1`. Finally, cases can be filtered according to some criteria, e.g. cases containing erroneous readouts or specific undesired DTCs should be removed from the data set, before we proceed with the actual analysis phase.

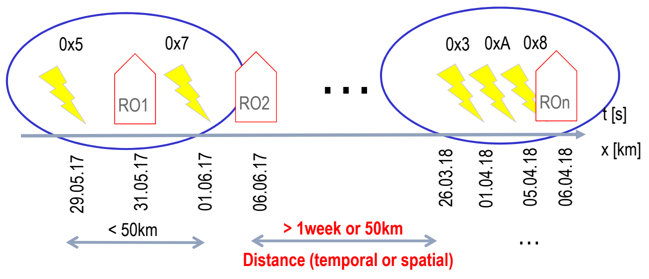


Fig. 1: Clustering in-vehicle DTC instances

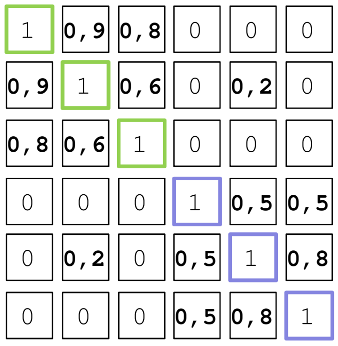
IV. Learning Patterns

Now, having obtained a large set of cases from the fleet of consideration, i.e. from a set of readouts $\mathcal{R}_t$, we want to find similarities and learn pattern classes from these cases. Unsupervised learning provides us with clustering techniques to group similar data samples into clusters. However, we first need to transform our cases to a convenient feature space with some appropriate distance metric in order to make them accessible for machine learning algorithms. A possible solution is to create a categorical vector to describe which DTCs are present in the case. Since there is an extremely large number of possible DTCs, we would get very high-dimensional but sparse vectors, since each vehicle only experiences a small number of DTCs. In order to cover the environmental conditions for each DTC, we would also need to extend the vector representation to a matrix. Each row in this matrix stands for a possible DTC while each column represents an environmental condition. Furthermore, there is one categorical column determining if the DTC is present at all. Having a $\mathbb{R}^{n\times m}$ dimensional feature space, where n is the number of DTC classes for the analyzed fleet and m is the number of environmental conditions, we can choose from a series of matrix norms to define a distance (or similarity) measure for clustering.

However, handling this explicit feature space with mostly extremely sparse samples is inefficient and also unnecessary. Instead we define a proximity measure on the concept of cases directly. Since a case is just a set of DTC instances, we can use set theoretic tools to obtain a proximity matrix, omitting a high-dimensional embedding of our data. We obtain the similarity $a_{1,2}$ of two given cases C_1 and C_2 using the Jaccard metric as

$$a_{1,2} = \frac{|\mathcal{D}_1 \cap \mathcal{D}_2|}{|\mathcal{D}_1 \cup \mathcal{D}_2|} \tag{4}$$

where $\mathcal{D}_1$ and $\mathcal{D}_2$ are the sets of DTC classes represented by the DTC instances of C_1 and C_2, respectively, i.e. $\mathcal{D}_i = class\,(C_i)$ if the $class$ function is applied element-wise. In other words, the similarity of two cases is the number of coinciding DTCs normalized by the total number of DTCs contained in these two cases. We thereby ignore the environmental conditions which turns out to not change the results much since the occurrence of a combination of particular DTCs carries much more information than the attached environmental parameters (however, environmental conditions can play an important role in pattern-based diagnosis; their integration into a pattern learning framework is subject of future work). Alternatively, each match in eq. (4) can be weighted by the similarity s_e of the corresponding environment vectors e_1 and e_2. The similarity should thereby be between 0 and 1, e.g. $s_e = e^{-||e_1 - e_2||}$ in order to keep the Jaccard index between 0 and 1 as well. The resulting similarity matrix is square symmetric as depicted in fig. 2. Values near to one denote a high similarity of the cases corresponding to the entry's row and column, respectively. Such cases are likely to end up in the same cluster.



1	0,9	0,8	0	0	0
0,9	1	0,6	0	0,2	0
0,8	0,6	1	0	0	0
0	0	0	1	0,5	0,5
0	0,2	0	0,5	1	0,8
0	0	0	0,5	0,8	1

Fig. 2: Similarity matrix produced by the Jaccard similarity metric for a set of six cases.

Further proximity metrics for categorical data can be found in [2]. However, we have learned from experiments that the recognized patterns barely depend on the choice of the similarity metric. Therefore, we omit a discussion on similarity metrics in this paper.

Unfortunately, many clustering algorithms cannot be fed with a proximity matrix directly. For instance, the popular k-means algorithm and its variations such as k-means++ create new data points (the cluster means) in the feature space during the clustering procedure [3]. This is not possible with our approach, as we do not have an explicit feature space and thus cannot provide a data matrix. However, algorithms such as spectral clustering and affinity propagation operate on similarity matrices directly and do not need an explicit high dimensional embedding [4], [5], [6]. Such algorithms are well-suited for our problem.

Although, unsupervised learning comes with the great advantage that we do not need any labels to learn a clustering, most algorithms require a small but important set of hyper-parameters directly or indirectly controlling the number of clusters to find. In our case the number of clusters is also the number of DTC patterns we want to find in our fleet. Of course, we cannot know this parameter in advance. However, we can estimate a reasonable value for the number of cluster by applying appropriate clustering evaluation measures. Since we want to keep the process completely automated, we are constrained to internal clustering indices such as Silhouette [7] and Calinski-Harabasz [8], i.e. indices not requiring labels to evaluate the quality of a clustering [9]. To obtain the optimal number of clusters with respect to an internal index, we re-compute the clustering for all possible cluster numbers from 2 to the number of cases under consideration. We then keep the clustering for which the internal index was optimal (maximal in most cases).

Such an exhaustive parameter search can take hours or days for fleets of a few thousands of vehicles on an ordinary computer. On the other hand, parallelization can be implemented easily, as all we need to do is distributing the clustering procedure to multiple computing nodes with different hyper-parameters. If computational resources are an issue, empirically supported heuristics can help. In our data sets, the optimal number of clusters was about two orders of magnitude smaller than the number of samples, e.g. having 10.000 cases, a good estimate is to extract 100 clusters.

V. Extracting Patterns

Having clustered similar cases into groups we are not yet finished. We now have to look into each cluster to find out what the core pattern of this cluster is. Again, each case is a set of DTC instances, but in order to use set theoretic tools we map the DTC instances to their corresponding classes, as was done for eq. (4). Figure 3 depicts two clusters as Venn diagrams with cluster 1 containing the cases 1-4 and cluster 2 containing cases 5 and 6. Mostly, the cases of one cluster are not completely overlapping, but tend to have large intersections. These intersections are re-occurring patterns and hence exactly what we are looking for. Non-overlapping areas on the other hand can be regarded as noise and probably have causes non-related to the main pattern of the cluster.

To find the main pattern in a given cluster, it is however not sufficient to compute the intersection of all cases of this cluster. Consider cluster 1 in fig. 3. The intersection of the four contained cases is a narrow stripe (filled blue area). However, if we remove case 4 from this cluster, we obtain the much larger striped intersection area (including the blue filled area). In this example, case 4 seems to be attached to cluster 1 rather by chance. This can happen if there is no better match for a case (like case 4) in the clustering phase. To get rid of such noisy candidates we define our pattern extraction procedure as follows: for each cluster find the largest set of DTCs present in at least $x\%$ of the cases related to this cluster. Thereby, x is another hyper-parameter which needs to be chosen carefully. Assume I is a set containing all case indices of a given cluster, e.g. for cluster 1 in our example $I = \{1, 2, 3, 4\}$. Then the pattern extraction is defined as the maximization problem

$$PC = \bigcap_{i \in S} \mathcal{D}_i, \tag{5}$$

with

$$S = \underset{\substack{S \in \mathcal{P}(I) \\ |S| \geq |I| \cdot x\%}}{\operatorname{argmax}} \left| \bigcap_{i \in S} \mathcal{D}_i \right| \tag{6}$$

where $\mathcal{P}$ is the power set operator.

Setting $x = 100$ means the pattern DTCs need to be present in *all* cases of the cluster. The extraction is then simplified to

$$PC = \bigcap_{i \in I} \mathcal{D}_i. \tag{7}$$

Finally, for each cluster, where the eq. (5) results in a non-empty set, we create a new pattern class. As discussed in section II, we interpret such a pattern class as a logical conjunction of DTC classes which is more convenient for further processing. According to this agreement the set notation $PC = \{\delta_1, \delta_2, \delta_3\}$ is semantically equivalent to the logical formula $PC = \delta_1 \land \delta_2 \land \delta_3$. For convenience of notation, we do not introduce two different notations to distinguish between the set theoretic and the Boolean logic syntax. The current interpretation should be clear from the algebraic context.

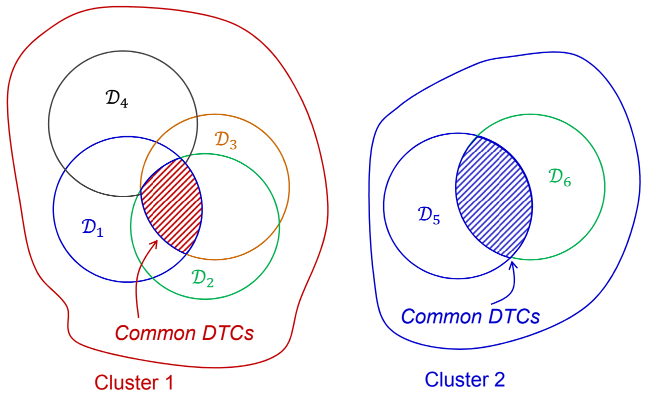


Fig. 3: Venn diagram depicting two clusters and their respective cases.

VI. Post-processing of error patterns

It turns out that many problems do not have a concrete set of symptoms, similar to medicine: if a patient has the flu, he is likely to have a fever. Additionally, he can experience nausea and/or limb pain. The possible pattern instances representing the flu then are
$\{fever\}$,
$\{fever, nausea\}$,
$\{fever, pain\}$,
$\{fever, nausea, pain\}$.

However, each patient can only have one of these four realizations (or pattern instances) of the flu at the same time. If we clustered patient data, we most likely would end up with four pattern classes for the flu, each describing one of the aforementioned pattern instances. However, these classes obviously share several commonalities and can be reduced to the two pattern classes $PC_{flu1} = fever$ and $PC_{flu2} = fever \land (nausea \lor pain)$. From the point of view of mathematical logic, PC_{flu1} implies PC_{flu2}, thus PC_{flu2} can be dropped. However, for diagnostics it is valuable to keep both patterns since, first, PC_{flu2} might increase the confidence of the diagnosis and, second, $PC_{flu1} \land \neg(pain \lor nausea)$ might turn out to be a symptom for another disease.

In the following we show how we combine pattern classes obtained in section V, which are exclusively conjunctions, to more complex patterns such as PC_{flu2}. We organize the remaining classes in a graph which helps an engineer to explore the pattern classes and their relationships.

First we order the pattern classes by the number of their atoms. Next, for each pattern class, starting with the shortest, we check whether it is a subset of another, longer pattern class. Thereby, we create an edge in an inclusion graph capturing the subset-of relationship for the two pattern classes. Consider the graph in fig. 4. Each box in the graph is a conjunctive pattern class as obtained from the pattern extraction step in section V. For the single-atom class a we find that there are two pattern classes implying a, namely $a \land b$ and $a \land c$. Therefore, these two pattern classes are connected with a. In turn, $a \land b$ is implied by $a \land b \land r$ and $a \land b \land v$ and so on.

Once the graph is complete we can compose nodes of the graph implying the same parent node. Thereby, the term of the parent remains a conjunction while the alternating end of the implying nodes is composed into a disjunction. In our example $a \land b$, $a \land c$, and $a \land d \land l$ all imply a. Hence, in the newly created composite we keep expression a as is (even if it is a more complex formula) and add a disjunction of the respective rests of the other three boxes, namely b, c, and $d \land l$. After this operation we always obtain two pattern classes: the parent pattern class and the newly composed pattern class, in this example a and $a \land (b \lor c \lor (d \land l))$. The parent class needs to be kept due to the reasons explained above.

Having applied the merging step to all nodes of the graph, we end up with a much more compact and expressive ordered list of pattern classes than the pure results of section V. The resulting pattern classes for the example are listed below the graph in fig. 4. Note that each line contains both the parent and the newly created child expression separated by a comma.

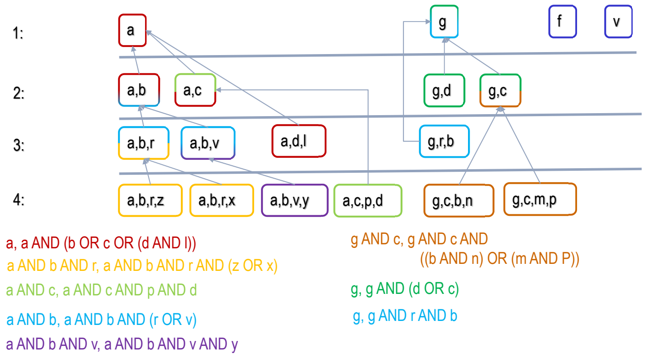

1:
2:
3:
4:
a
a,b
a,c
a,b,r
a,b,v
a,d,l
a,b,r,z
a,b,r,x
a,b,v,y
a,c,p,d
g
f
v
g,d
g,c
g,r,b
g,c,b,n
g,c,m,p
a, a AND (b OR c OR (d AND l))
a AND b AND r, a AND b AND r AND (z OR x)
a AND c, a AND c AND p AND d
a AND b, a AND b AND (r OR v)
a AND b AND v, a AND b AND v AND y
g AND c, g AND c AND ((b AND n) OR (m AND P))
g, g AND (d OR c)
g, g AND r AND b

Fig. 4: Organized error pattern tree

```
1 (declare-const x1 Bool)
2 (declare-const x2 Bool)
3 (define-fun  patterneq () Bool
4   (= (not (and x1 x2))  (or (not x1) (not x2))))
5 (assert (not patterneq))
6 (check-sat)
```

Fig. 5: Generated z3 SMT solver code to semantically compare two error patterns.

VII. Storing and Extending the Results

Note that semantically equal Boolean expressions may be written in completely different ways. Imagine, we have found two syntactically different DTC pattern classes $PC_1 = \neg(x_1 \wedge x_2)$ and $PC_2 = \neg x_1 \vee \neg x_2$ from two independent clusters. From De Morgan' laws we know that these two expressions are equal for all Boolean assignments of x_1 and x_2. It is crucial to find such duplicates in order to keep the pattern database clean and avoid confusions.

This can be done automatically by employing the z3 SMT solver [10]. For each pair of pattern classes we generate a z3 script containing a definition of a Boolean variable for each DTC class contained by the two pattern classes, c.f. lines 1-2 in fig. 5. Next we define a function equating the Boolean expressions of the two pattern classes in lines 3-4. Finally we declare the assertion that the above defined function is evaluated to false in line 5. The model is checked in line 6. As there is no variable assignment satisfying the assertion, z3 reports that the model is not satisfiable. Thus, we know that the two patterns are equal. Consequently we remove one of these patterns from the list.

This procedure not only helps to remove duplicates from the analysis results, but also ensures that no redundant information is persisted in the pattern database. As described in section V, the analysis process takes place on a regular basis. Analysis results obtained on a dataset at t_0 are stored in the pattern database. The following analysis at a later point in time t_1 might find some of the previously found patterns again. Therefore, in the z3-based reduction step described above we include not only the currently found pattern classes, but also all previously found and manually created ones.

To provide an up-to-date predictive maintenance or to be able to analyze new DTC patterns as soon as they start emerging in customer vehicles, our analysis process needs to be run on a regular basis. Thereby, two questions arise: what data should be included in an analysis and how should the analysis frequency be chosen? It is practically a tautology in machine learning that the more data we have, the better the results. This means we should include all available data to detect the most relevant patterns with high probability. However, this is a fallacy in our problem setting. Including all historical readouts in the analysis dataset would lead to favoring old and therefore already known patterns as they have most probably occurred in far more vehicles than new DTC patterns caused by a recent software update. Consequently, the detection of new patterns is delayed to a point in time when they outweigh older patterns (which might never occur). Therefore, we recommend using only readouts recorded since the last analysis, i.e. instead of using the set of readout $\mathcal{R}_t$ at time t we only consider the set difference $\mathcal{R}_t \setminus \mathcal{R}_\tau$ where τ is the last known analysis time.

This brings us to the second question concerning the analysis frequency which turns out to be an important hyper-parameter. Similar to signal processing and spectral analysis, we have to cope with the so called uncertainty principle, also referred to as the Gabor limit [11]. The more often we run our analysis, the more up-to-date and topical the DTCs our analysis sees are. However, the higher the analysis frequency, the less representative the sample becomes of the underlying distribution. On the other hand, a low analysis frequency leads to the aforementioned blurring of patterns from different time periods and long recognition latencies. In practice, the analysis frequency highly depends on the fleet size: the larger the fleet, the more DTCs are recorded and the less time is needed to gather a representative amount of data. Another important factor is the technology used to collect readouts: in the past, OEMs only obtained DTC data read by themselves or by maintenance shops using a diagnosis device. Today, DTC data can be sent using the vehicle's radio to the OEM as soon as it is produced. This ensures that all DTCs data arrives at the database of the OEM while the delay of waiting for the vehicle to be checked at the shop is removed, allowing a higher analysis frequency.

VIII. Evaluation

The proposed approach was evaluated on DTC data covering approximately a one year period and a fleet size of the magnitude of 10.000 vehicles with at least one readout. The obtained patterns were compared with the internal database of the corresponding OEM. Unfortunately, neither the datasets, nor the patterns can be made public due to confidentiality. However, our main findings were that most DTC pattern classes present in the internal pattern database created by human engineers where found by our approach as long as these patterns affected some critical number of vehicles. The algorithm even managed to find single DTC patterns which we had assumed to be difficult to detect. Furthermore, the algorithm was able to find patterns which were not present in the database, but which, according to domain experts, were definitely important and should be captured for future use.

IX. Conclusion

In this paper we have presented an approach to fully automated and unsupervised error pattern learning from DTC data using clustering techniques combined with Boolean logic and set algebraic tools. The approach reduces human effort and time needed to find and formalize patterns, but also ensures that domain knowledge is always made explicit. Furthermore, re-occurring patterns are easily detected and duplicates are eliminated using satisfiability solvers. The extracted patterns can be used in classical quality assurance processes of vehicle OEMs, but also serve as an important basis for predictive maintenance and automated on-line self-diagnosis in intelligent vehicles.

Future work comprises an extension of the proposed analysis process by supervised learning techniques in order to train a system to automatically suggest a repair solution for a given error pattern. To facilitate research in this area, we encourage OEMs to make chunks of OBD and repair data publicly available.

References

[1] John A Hartigan. Consistency of single linkage for high-density clusters. *Journal of the American Statistical Association*, 76(374):388–394, 1981.

[2] Shyam Boriah, Varun Chandola, and Vipin Kumar. Similarity measures for categorical data: A comparative evaluation. In *Proceedings of the 2008 SIAM International Conference on Data Mining*, pages 243–254. SIAM, 2008.

[3] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[4] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[5] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[6] Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[7] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[8] Tadeusz Caliński and Jerzy Harabasz. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27, 1974.

[9] Yanchi Liu, Zhongmou Li, Hui Xiong, Xuedong Gao, and Junjie Wu. Understanding of internal clustering validation measures. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 911–916. IEEE, 2010.

[10] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[11] Karlheinz Gröchenig and Georg Zimmermann. Hardy's theorem and the short-time fourier transform of schwartz functions. *Journal of the London Mathematical Society*, 63(1):205–214, 2001.