Design and optimization of an autonomous feature selection pipeline for high dimensional, heterogeneous feature spaces

Bernhard Schlegel
Bayerische Motoren Werke AG
Munich, Germany
Email: bernhard.bb.schlegel@bmw.de

Bernhard Sick
University of Kassel
Intelligent Embedded Systems
Kassel, Germany
Email: bsick@uni-kassel.de

***Abstract*—The growing complexity and diversity of vehicle systems makes it increasingly hard to identify and resolve the root cause of an unplanned maintenance session at hand in a dealer workshop. This especially holds for workshop staff with limited qualifications and experience. By today, the workshop staff is supported by expert-based systems, where knowledge was manually generated, i. e., by formalizing human knowledge using rules. Due to the above mentioned reasons, this approach is coming to its limits while the potential of car- and workshop-data available today remains unused. A highly autonomous, machine learning based approach seems promising. To unleash its full potential, the selection of relevant features from the multi-thousand dimensional feature space is indispensable. This feature selection needs to take the automotive requirements into account. These include, but are not limited to, sparse data, the requirement that the selected features are interpretable, and the aim that the trained models save warranty costs. This article presents and evaluates a hands-free feature selection pipeline, paving the way for automotive model building. The pipeline consists of three layers: a feature preparation layer, a filter layer that significantly reduces the feature space at low computational cost based on entropy and statistical measures, and a wrapper layer that selects the final feature set for training based on simple models. Finally, highly performant logistic regression models have been trained to generate the metrics used for evaluation.**

I. Introduction

Misdiagnosed and misrepaired cars are an important cost factor in the automotive field[1] and influence the brand customers' perception negatively. The current expert knowledge based offboard diagnostics practiced in workshops are not able to cope with a growing variant diversity, differing environmental conditions, and the increasing product complexity. Because of hybrid cars integrating two drive trains, this trend is expected to continue.

The shift from expert knowledge based diagnostic systems to data-driven systems utilizing the available data is a promising approach. To cope with the vast amount of data produced by cars and workshops nowawdays, relevant features in a multi-thousand-dimensional feature space must be identified. This becomes additionally challenging when the features vary in their sparsity, noise level, and distribution.

[1]The warranty cost of BMW increased by 30% to 1.89 Billion from 2014 to 2015. [1]

To pave the way for a highly automated, machine learning based diagnostic approach, a feature selection pipeline is needed since 5000 features or more are not uncommon in an automotive context. Often, as our results show, no more than 25 features are needed for modelling. Therefore, the pipeline must be able to reduce the high-dimensional feature space by up to 99.5%, taking into account the automotive application requirements while ensuring maximum interpretability of the produced results.

This article focuses on feature subset selection techniques to ensure the interpretability of the selected features by humans. Dimension reduction techniques that transform features into another (meta-)feature space (e.g., principal component analysis [2], deep learning) or compress features based on information theory are, therefore, not part of this research [3].

The contribution of this article is the implementation, systematic evaluation, and optimization of a highly autonomous feature selection pipeline for high dimensional feature spaces. Applications include, but are not limited to, offboard car-diagnostics to assist workshop staff by predicting faulty parts or suitable countermeasures and predictive maintenance. This leads to three main topics that define the structure of this article which is outlined below, after related work was examined (Section II) and the data sets upon which this article is based are introduced (Section III).

First, a formal description and evaluation of the existing features and potential model outputs (targets) is needed (Section IV). Second, the basic structure of the pipeline has to be defined incorporating the aforementioned requirements (Section V). Third, a multidimensional optimization problem needs to be solved: How to identify the optimal hyperparameters and algorithms for the pipeline and what are the most valuable features? (Section VI)

The feature evaluation (Section IV) deals with the question how the existing feature groups can be described and classified.

In the second part (Section V) we will answer the question, how to chain available methods and algorithms in the most promising way in a pipeline. This incorporates the following questions: Which characteristics have to be obeyed when

preparing the features for the pipeline (Section V-A)? Which filter measures (Section V-B) and wrapper algorithms (Section V-C) should be evaluated? Which kind of model is suitable for generating a pipeline performance metric, used for evaluation and optimization (Section V-D)? And at last: Which parameter values can be used for optimizing the pipeline (Section V-E)?

In Section VI we analyse thousands of results to identify the optimum hyperparameters of the pipeline and prove that it is possible, given the available features, to create models that produce results above a given performance threshold. In a detailed evaluation we will address the question, which filter algorithms rank features high in order to create highly performant models (Section VI-A). Furthermore, a variety of wrapper algorithms (Section VI-B), the pipeline, the remaining hyperparameters (Section VI-C), and the different feature groups are evaluated. Moreover: How does the proposed pipeline perform on data sets for which it has not been optimized (Section VI-E)?

Finally, we sum up the gained insights and give a recommendation how to adapt the proposed pipeline and ideas to other environments (Section VII), draw our conclusions, and give an outlook (Section VIII).

II. Related Work

Most known data-driven approaches from automotive contexts based on machine learning are *specific* in the sense that they only diagnose a single type of component. These components include, e.g., compressors, bearings, combustion engines, or turbochargers [4]–[8].

Generic approaches are able to predict different types of faults: Azarian et al. [9] propose a generic approach that is only evaluated on three cars and requires high manual effort. Another generic approach is proposed by Müller et al. [10]. Their model is based on

- Diagnostic Trouble Codes, *DTCs* (Based on the self-diagnostic capabilities of electrical control units (*ECU*s), e.g., discrepancies between measured and calculated values can be flagged as a DTC.)
- encoded customer and workshop staff perception,
- software versionnumbers of *ECU*s, and
- part numbers of switched parts.

In contrast, the pipeline proposed in this article is developed and evaluated on the following feature groups

- long time measurement values, *MV* (Captured values of slowly changing car parameters, e.g., adaption of proportional-integral-derivative controllers integrated into the engine control.),
- environmental conditions from DTCs (Every time a DTC is flagged, environmental conditions such as the engine rounds per minute are captured.),
- optional extra equipment (For example, if the vehicle is equipped with leather seats or an aerodynamic kit.),
- basic car parameters (This information includes, e.g., the engine power, steering type, etc.), and
- basic readout data (A readout contains, e.g., the current odometer value.)

III. The Data Sets

The pipeline was developed based on the characteristics of automotive workshop and car data (Section III-A). To ensure comparability, the proposed pipeline was additionally evaluated on publicly available data sets (Section III-B).

A. Workshop and car data

The *automotive* data set was collected from BMW Hybrid cars and consists of non personal data only. In total, 80.000 observations, each with 5000 features were used to predict over 3000 potential targets. These include the switched parts, taken actions, and diagnostic codes. These codes are hash-like values identifying the final result of the workshop repair. A detailed evaluation is given in Section IV.

B. Publicly available data sets

Two publicly available data sets were used: The *golub* (where the relevant features can be easily to figured out) [11], and the *secom* data set [12]. Both data sets can be obtained online and have a binary target variable.

The *golub* data set contains 72 observations with 7129 features each. The features in this case are gene expressions. The target is to classify which of these observations indicate type one leukemia (47) or type two (25).

Secom contains 1567 observations from a semi-conductor manufacturing process. The 591 features represent sensor values used for monitoring the process. The goal in this case is to classify a "pass" of the factored product, which is recorded 1463 times. A "fail" is recorded 104 times.

IV. Evaluation of the automotive data

Since dealing with automotive data is the main goal of this article, a detailed analysis was done prior to model building. Each of the feature groups in the data set has its own properties, making the feature selection process difficult. Extra equipment is boolean (a car can either be equipped with leather seats or not), car parameters are mostly categorical (a car engine is only available at discrete performance levels), and mostly all other parameters are numerical or interval scaled after being preprocessed (e.g., the mileage of the readout, the value of a specific MV, etc.). Additionally, the sparsity differs between the different feature groups: DTCs have a sparsity of 99%, whereas MVs only have a sparsity of 65% and readout data less than 1%.

Many potential targets only appear at a very low rate. We set the minimum number of observations per target to five. This minimum is only exceeded by 69% of the parts, 56% of the taken actions, and 40% of the diagnostic codes. Only targets that matched this criterion (e.g., a part that was switched at least five times) have been processed by the pipeline.

V. Feature Selection Pipeline

The complete pipeline (see Figure 1) consists of preparation layer (Section V-A), two feature selection layers, and a model building layer (Section V-D), all implemented in R. The feature selection layers include a filter layer (Section V-B) and a wrapper layer (Section V-C).

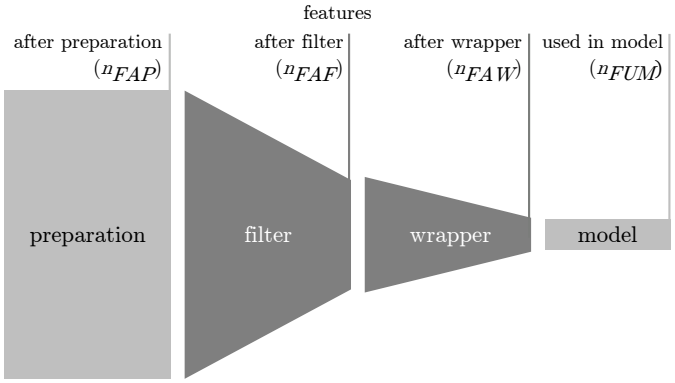

Fig. 1. Overview of the feature selection pipeline.

A. Preparation layer

To ensure maximized automation, this layer for data preparation is restricted to methods that do not, once implemented, require any kind of manual intervention (*parameter free*). Preparation involves removing constants, features that have a linear correlation with other features, and coding textual features (e.g., the software-version of the car which is encoded like "BMW3-16-13-100") into numbers.

Also, an automotive context suited outlier filter is implemented for numeric values: Values that lie within the first $q_{0.01}$ or last $q_{0.99}$ percentile are set to the closest, valid value. This is necessary because standard outlier filters based on mean and standard deviation end up in classifying too many values as outliers due to the non-normal distribution of the data. E.g., in Figure 2 all values on the right of the dashed line would be wrongly classified as outliers. In this way, two goals are accomplished: First, a broad range of values is kept. Second, values initialized with extreme values (e.g., 65536) do not shrink all other feature values (e.g., normally distributed with $\mu = 1000$ and $\sigma = 100$) when being centered (the features mean is subtracted) and scaled (the centered values are divided by the features standard deviation) prior to model training. This results in a feature with $\mu(feature) = 0$ and $\sigma(feature) = 1$.

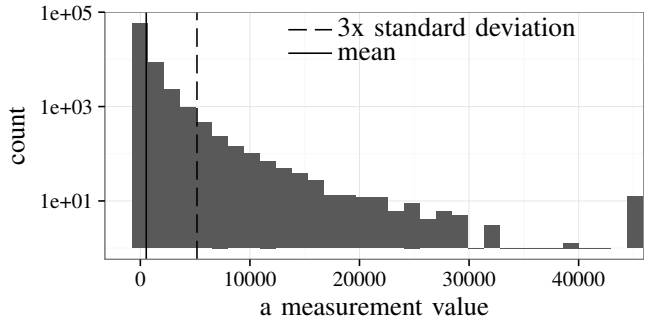

Fig. 2. Example why standard outlier filters fail.

Furthermore, a variety of nominal features exists that are ordinally encoded. So, although being numerical per se, they are not linearly interpretable. Take, e.g., the dealer number referencing all BMW branches and workshops. Although being numerical, a dealer referenced by 21788 may reside on a different continent compared to a dealer that is only "1" unit away. A translation to latitude and longitude coordinates fixes this problem in most cases. Through these coordinates a linear connection is established, making the geographical information (e.g., dusty areas or areas with low gasoline quality) accessible to the model.

After this step, the data are split into a training/validation and a test set. All following steps that reduce the number of dimensions are only conducted on the training/validation sets.

B. Filter layer

Filters are fast, scalable, and independent from the classifier [3]. Therefore, the filter layer is a crucial building block in the pipeline, reducing the dimensionality at low computational cost. The filter layer evaluates the inputted features using multiple, differing measures:

- *Correlation*: Measures the correlation between the feature and the target variable.
- *Chi-squared,* χ^2: Measures the similarity of the feature and the target distribution.
- *Gini coefficient*: Measures the inequality for the features frequency distribution.
- *Information gain*, IG: Entropy H based measure holding the mutual information of the feature and the target. Calculated using $IG_{feature} = H(Class) + H(feature) - H(Class, feature)$ – unlike the IG from random forests.
- *Relief*: Estimates features based on how well their values can be used to distinguish instances that are close to each other in the feature space [13].

The final ranking is calculated "parameter-free" for every feature by summing up all filter measure rankings, each normalized to a range from 0 to 1. This is repeated for every model, each representing one potential output. Also, a "take top N approach" is evaluated. In this mode, the N top performing features from each filter measure are selected, followed by the features ranked high by the sum over all algorithms. For this article, the values of N are set to 0 and 1, respectively.

A disadvantage of filters is that they ignore feature dependencies as well as the classifier dependent feature preferences, e.g., the total number of features used to create a classifier. Therefore, as we will show later, the optimal pipeline selects as many usefull features as possible from the filter layer, reducing the risk of eliminating precious features and leaving it up to the wrapper to choose the features that are really worth being kept.

C. Wrapper layer

Once the feature space has been "filtered" at low computational cost, the wrapper layer evaluates the remaining features in greater detail. Three concepts were implemented and evaluated: The information gain based ranking of a random-forest (*RF*) wrapper, the feature-weight based ranking of a logistic

regression (*LR*) wrapper, and a k-nearest neighbors (*KNN*) greedy forward selection [14] [15] embedded approach.

An embedded approach performs feature selection as a part of the learning procedure (e.g., *RF* and *LR*). A wrapper approach, on the other hand, performs the selection by evaluating the model-performance on different feature subsets of the features passed from the filter layer (e.g., *KNN*) [14]. For simplification, both approaches (wrapper and embedded) will be referenced as "wrappers" due to their similarity.

The RF wrapper uses the random forest provided by the `caret` package, available for `R`. The feature ranking is calculated using the averaged position in the trained trees (the forest). This way, feature dependencies are taken into account. The position depends on the feature specific information gain IG (the bigger the IG, the higher the feature is located in the tree). In this case, IG is calculated according to [2] using the entropy H before and after the tree was split into two branches by the evaluated feature:

$$\begin{aligned} IG &= H_{before} - H_{after} \\ &= H_{before} - (H_{leftBranch} + H_{rightBranch}), \text{ where} \\ H &= -\frac{n_p}{n_t} * \log_2(\frac{n_p}{n_t}) - \frac{n_n}{n_t} * \log_2(\frac{n_n}{n_t}), \\ n_t &= n_p + n_n, \end{aligned}$$

$n_{p/n}$: number of positive/negative observations.

The LR wrapper uses the feature weights after training has finished, considering feature dependencies.

The greedy KNN embedded approach calculates the KNN model accuracy for every feature one by one. After every iteration, the feature is added to the final ranking and removed from the list that contains the features which still have to be evaluated. In this way, feature dependencies are not taken into account.

D. Model layer

To generate the metrics used for evaluating the pipeline, a L1-regularized LR model was used. We used the `LiblineaR` implementation – a wrapper for the `LIBLINEAR C/C++` library [16]. During training, the following unconstrained optimization problem is solved for all instance-label pairs $(\mathbf{x}_i, y_i), i = 1, ..., l, \mathbf{x}_i \in R^n, y_i \in \{-1, +1\}$:

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi(\mathbf{w}; \mathbf{x}_i, y_i)$$

For each model, seven different costvalues C are evaluated and optimized (ranging from 10^3 to 10^{-3}) with regard to the area under the Receiver Operating Characteristic curve (*ROC*, see Section V-E) using a 5-fold cross validation. C indicates the trade-off between regularization (shrinking $\mathbf{w}$) and correct classification (minimizing the loss function ξ). The loss function is defined as

$$\xi(\mathbf{w}; \mathbf{x}_i, y_i) = \log(1 + e^{-y_i\mathbf{w}^T\mathbf{x}_i})$$

The LR was used, because the output of a "predicted probability" (ranging from 0 to 1) enables an in-depth evaluation. Also, the computational complexity is relatively low.

E. Defining the optimization criteria

In order to be able to compare the various methods and algorithms described in the following, an objective performance measure for the LR model is needed. A scalar value has practical advantages by allowing for an easy comparison.

All measures calculated from a confusion matrix such as accuracy, precision, recall, F1 score (F_1), etc. disregard the predicted probability by setting a fixed threshold[2] ignoring valuable information. In contrast, the ROC is a 2D curve taking this information into account. The same applies to a precision-recall curve, but the precision is dependent on the positive to negative sampling rate, which is varied and evaluated.

This predicted probability corresponds to the predicted value of the LR model: An output of 0.99 meaning the model considers a result of 1 to be very likely. If the output is 0.63 the model is definitely less "sure" what the output will really be. For detailed information how to draw an ROC curve based on these "predicted probabilities", we refer to [17].

Integrating the ROC curve results in an one-dimensional characteristic, the area under the ROC curve or *AUC*. It can be empirically and formally shown that AUC is a better measure than the widely used accuracy [18] [19]. In addition, the AUC is enhanced by the corresponding F_1 in the following.

VI. Evaluation and Optimization of the Pipeline

The search grid is spanned up by systematically varying each of the following hyperparameters, defining the shape and behaviour of the pipeline:

- n_{FAF}: The number of features left after the filter layer (25, 50, 100, 200, 500).
- n_{FAW}: The number of features left after the wrapper/ embedded layer (5, 10, 15, 25, 50, 500).
- NPR: The positive to negative observations sample ratio defines how many negative observations (e.g., part was not switched) are sampled from the data set depending on the number of available, positive (e.g., part was switched) observations (1, 3 or 5 times).
- $SPLIT$: E.g., 0.7 meaning 70% of the data is used for training. Evaluated values are 0.7 and 0.8.
- WA: The used wrapper algorithm – either *KNN*, *RF*, or *LR*.
- TFA: Use the top ranked feature from each filter measure in any case (1) or select features based on the summed ranking over all filter measures (0).

This yields $n_{sp} = 5 * 6 * 3 * 2 * 3 * 2 = 1080$ sampling points. E.g., one being defined as $n_{FAF} = 200$, $n_{FAW} = 15$, $NPR = 3$, $SPLIT = 0.8$, $WA = KNN$ and $TFA = 1$. Every point is sampled 10 times for smoothing. Multiplied by the total number of successfully evaluated targets (832) this yields $n_{trainedModels} = n_{sp} * 10 * 832 = 8.986 * 10^6$. For each sampling point, minimum, maximum, standard deviation (σ), mean, and median of AUC and F_1 are recorded.

[2]For example, if the threshold is 0.5, all predicted probabilities ≥ 0.5 will be considered positive for the confusion matrix.

The pipeline can only be evaluated[3] if the available features allow for a creation of models at all. Only potential targets with five or more observations have been processed. This only holds true for $\approx 35\%$ of all potential targets. Figure 3 shows the percentage of models for the remaining targets that performed above a given average AUC and F_1 depending on the target type. An explanation for the partially low AUC and F_1 is that some potential targets are too generic. Take, e.g., a bolt (switched part) that is used in a variety of different situations, the clearing of all DTCs (a taken action) which is common after a fault has been found and fixed, or a planned routine maintenance (diagnostic code) that has no unique triggers other than an approximate mileage or other specified usage.

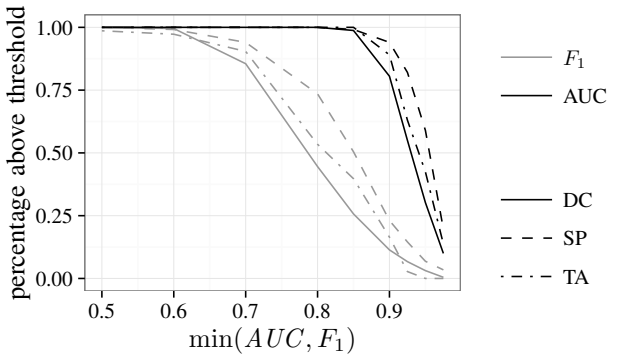

Fig. 3. Average model performance depending on target type.

The following sections examine the effect of the pipeline hyperparameters, evaluate the algorithms, and give a measure of the overall pipeline performance.

A. Filter Layer Evaluation

The implemented filter measures are evaluated to identify those measures that positively affect the performance of the pipeline. On the other hand, measures that only contribute little to the overall classification performance but significantly increase the computational complexity can be eliminated. This way, the pipeline is optimized for future research.

To rank the filter measures, more than 39000 filter rankings have been matched with the resulting models. The results are listed in Table I.

Every time, a filter measure ranks a feature higher than $w_{min} = 0.001$ that is used in the model with an absolute feature weight higher than w_{min}, a hit is counted. The column "% hit" in Table I is the number of hits (every hit counts as 1) from each measure divided by the number of features with an absolute weight higher than w_{min}, in the following referred to as n_{GT0}, used across all trained models.

Therefore, a high "hit percentage" does not necessarily imply a useful filter measure: A filter measure ranking all features passed to the next layer with an "importance" ranking of 0.01 would score 100%.

In addition to counting the number of "hits", the "contribution" column takes the filter measure ranking and the feature weight of the model into account. In case of a "hit", the filter measures importance ranking is multiplied by the feature weight in model. This number is summed over all models and divided by n_{GT0}.

The measure Relief is unreasonable to use, given it's extraordinary high computational cost ($runtime = 13.722s$, see Table I), and the fact that its contribution turned out to be about the half of the χ^2-measure. Also, the results of χ^2 and the information gain based filter yield the exact same results regarding the hit percentage. Therefore, the information gain measure may be dropped due to its higher computational complexity ($runtime = 2.381s$) compared to the χ^2 measure ($runtime = 1.916s$) if the computational complexity needs to be reduced.

TABLE I
FILTER MEASURE PERFORMANCE.

measure	% hit	contribution	runtime [s]
Relief	0.961%	0.466	13.722
Information gain	0.829%	0.763	2.381
Correlation	0.988%	0.671	0.262
χ^2	0.829%	0.890	1.916
Gini	0.989%	0.674	0.243

TABLE II
INFLUENCE OF THE WRAPPER ALGORITM ON THE PIPELINE PERFORMANCE.

algorithm	AUC	F_1	runtime [s]
KNN	0.923	0.771	27.851
LR	0.948	0.836	0.258
RF	0.928	0.791	10.710

B. Wrapper Layer Evaluation

Three different wrapper algorithms were evaluated (hyperparameter *WA*): A greedy KNN, RF, and an LR (Table II). The KNN and RF wrapper algorithms yield comparable results in terms of model performance ($AUC_{RF,average} = 0.928$, $AUC_{KNN,average} = 0.923$), only the LR wrapper yields slightly higher AUC and F_1 scores. This may be due to the fact that the model is also an LR. Both, the RF and especially LR wrapper algorithms are way less computationally expensive compared to the KNN approach.

Figure 4 shows the number of actually used features by the model n_{FUM} depending on n_{FAW}. Especially for high thresholds, where a feature is only counted if its feature weight w[4] in the final model exceeds threshold $t \geq 1$, n_{FUM} starts decreasing after more than $n_{FAW} \approx 30$ features are passed by the model. The number of selected features with threshold higher than 0.1 increases monotonically. Therefore, at least 30 features should be passed to the final model.

[3]We used a HP Z840 workstation equipped with two Xeon E5-2640 CPUs and 96GB of RAM.

[4]Averaged across all trained models, w was defined by the following metrics: $mean = 0$, $median = 0$, $\sigma = 2.35$, $min = -603$, and $max = 452$.

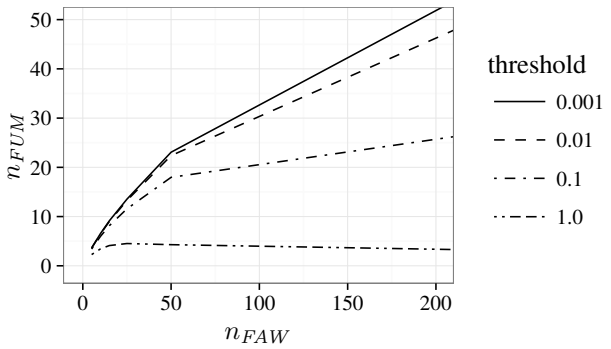

Fig. 4. Number of features used in final model above threshold.

TABLE III
INFLUENCE OF n_{FAF} ON PIPELINE PERFORMANCE.

n_{FAF}	AUC	F_1
25	0.928	0.784
50	0.934	0.799
100	0.934	0.801
200	0.934	0.802
500	0.934	0.806

TABLE IV
INFLUENCE OF n_{FAW} ON MODEL PERFORMANCE AND TRAINING TIME.

n_{FAW}	AUC	F_1	average training time [s]
5	0.906	0.722	0.105
10	0.928	0.786	0.143
15	0.936	0.810	0.166
25	0.944	0.831	0.177
50	0.953	0.847	0.159
500	0.951	0.845	0.255

C. Remaining hyperparameters

The influence of n_{FAF} on the pipeline's performance is given in Table III. For n_{FAW} we refer to Table IV. Both, n_{FAF} and n_{FAW} positively correlate with the pipeline's performance ($corr(n_{FAF}, AUC) = 0.011$, $corr(n_{FAW}, AUC) = 0.064$). This means, the more features are passed from the filter layer over the wrapper layer to the model layer, the higher the performance (in terms of AUC and F_1).

Although a higher n_{FAW} leads to a higher model performance, the model training time increases disproportionately: While the AUC increases by only 5.2% (F_1 increases by $17,31\%$) when increasing the number of features used by the model from 5 to 50, the model training time increases by 51.4%. Since the increase of training time is 10 times (four times) as high as the increase of the AUC (F_1), the optimum number of features passed from the wrapper layer to the model is set implicitly by the desired model performance in terms of AUC and F_1.

The next hyperparameter we evaluated is NPR. According to the results in Table V, the AUC increases with a higher

TABLE V
INFLUENCES OF *NPR* ON THE PIPELINE PERFORMANCE AND TRAINING TIME.

NPR	AUC	F_1	model training time [s]
1	0.919	0.878	0.076
3	0.938	0.796	0.139
5	0.942	0.724	0.245

NPR. A reason, why F_1 does not increase as well, can be explained by the way F_1 is calculated [2]:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

It represents the *harmonic mean* of $precision$ and $recall$. Thus, a lower $precision$ always causes a lower F_1 if the $recall$ remains the same. $Precision$ and $recall$ are calculated as shown below:

$$precision = \frac{TP}{TP + FP} \qquad recall = \frac{TP}{TP + FN}$$

TP : true positives $\qquad$ FN : false negatives

FP : false positives

An increase of the number of negative samples in the used data set will always lead to a higher number of FPs under the assumption that the FP to *true negative* ratio is constant. This leads to a smaller $precision$ while the $recall$ is not dependent on the number of negative samples. Therefore, increasing the number of negative observations in the used data set, will always lead to a smaller or equal F_1.

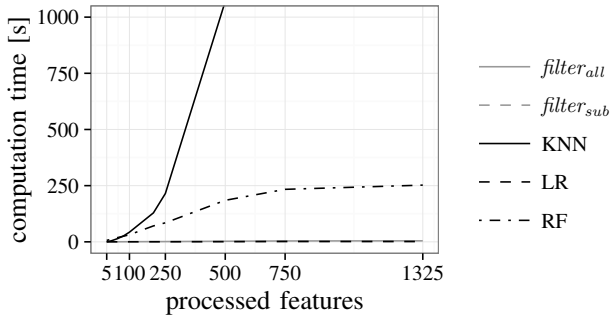

Fig. 5. Computation time: KNN, RF and the filter layer.

We achieved higher results with a $SPLIT$ of 0.8, independent of the number of available positive observations. The influence of TFA can be neglected. This can be explained by the fact that a feature that is ranked high by a specific filter measure will be very likely part of the of feature set passed to the wrapper layer if 20 or even 200 features are selected.

Aside from evaluating the given hyperparameters, we also addressed computational complexity and scalability of the proposed pipeline. Figure 5 shows the computation time depending on the number of processed features. For the numerical values, we refer to Table VI. To generate the values for column $filter_{all}$, all measures introduced in Section V-B

have been used. For two reasons we introduced a subset of filter measures: First, some filter measures are unreasonable to use as shown in Section VI-A. Second, an LR wrapper is computationally very cheap. Column $filter_{sub}$ in Table VI therefore only includes the correlation and Gini measures and computes faster than the LR wrapper.

Because of an computation time almost independent on the number of processed features, both filter layer configurations and the LR wrapper algorithm are considered absolutely scalable. The RF wrapper algorithm does also scale, but takes longer to compute. In contrast, the computation time of the KNN algorithm grows disproportionately.

To approximate the time savings gained by using the proposed pipeline, we use the following, conservative formula (where 100 features are processed by the RF wrapper). We compare the processing time of a standalone wrapper approach to a combined filter/wrapper approach. The number of features inputted does not exceed $n_{unfiltered} = 1325$ in this example. This limitation may not hold in future or other applications, where the time savings would be even greater:

$$\begin{aligned} t_{saving,RF} &= t_{RF,unfiltered} - (t_{filter_{sub},unfiltered} + t_{RF,100}) \\ &= 252.433s - (4.460s + 31.613s) \\ &= 216.307s \end{aligned}$$

which results in a saving of

$$t_{saving,RF} / t_{RF,unfiltered} = 85.689\%$$

As shown in Table VI, there are combinations where no time is saved, though. E.g., if the filter layer utilizes all filter measures and the wrapper layer is using an LR:

$$\begin{aligned} t_{saving,LR} &= t_{LR,unfiltered} - (t_{filter_{all},unfiltered} + t_{LR,100}) \\ &= 1.550s - (4.460s + 0.267s) \\ &= -3.177s \end{aligned}$$

Only the use of a subset of filter measures (e.g., $filter_{sub}$) in a pipeline incorporating an LR wrapper causes a decrease in terms of runtime.

D. Importance of feature groups

Evaluating all successfully trained models, the used feature groups (FG) are evaluated in Table VII. The row "Bias" references the linear offset of the LR model, being part of all created models. The column "used" shows a mixture of features used averaged across all models:

$$used_{FG} = \frac{\sum_{m=1}^{M} \sum_{f=1}^{F_m} \mathcal{C}(f, FG)}{\sum_{m=1}^{M} F_m}, \text{ with}$$

$$\mathcal{C}(f, FG) = \begin{cases} 1, & \text{if } f \text{ is a feature of } FG \\ 0, & \text{otherwise} \end{cases}$$

M : number of models,

F_m : number of features in model m.

Whereby "used" only considers if a feature from the corresponding group is used or not, the "weighted" column also takes into account the feature weight, used by the L1 regularized LR model. The summed feature weight of each feature group is divided by the total weight of all features used by the model. Again, this number is averaged across all models for all FGs:

$$weighted_{FG} = \frac{\sum_{m=1}^{M} \sum_{f=1}^{F_m} \mathcal{W}(f, FG)}{\sum_{m=1}^{M} \sum_{f=1}^{F_m} \sum_{g}^{G} \mathcal{W}(f, g)}, \text{ with}$$

$$\mathcal{W}(f, FG) = \begin{cases} w_{m,f}, & \text{if } f \text{ is a feature of } FG \\ 0, & \text{otherwise} \end{cases}$$

$w_{m,f}$: weight of feature f in model m,

G : number of feature groups.

Especially MVs turned out to be very helpful in the model creation process; they have not been used in prior work [10]. The surprisingly low impact of DTCs can be explained by the complex set of conditions by which they are triggered: Often, multiple (even timed) conditions are joined by different logical operators (`OR`, `AND`, `XOR`, etc.) to flag a DTC. Despite this wealth of information that is needed to flag a DTC, the way back from a DTC to the conditions upon which it was triggered is surjective and not bijective - different sets of conditions can cause the same DTC.

TABLE VI
OVERVIEW OVER THE LAYER RUNTIMES IN SECONDS DEPENDING ON THE NUMBER OF PROCESSED FEATURES.

$n_{feature}$	$filter_{all}$	$filter_{sub}$	KNN	LR	RF
5	0.043	0.007	0.740	0.103	6.957
10	0.060	0.007	1.320	0.100	7.797
20	0.100	0.040	3.517	0.140	10.393
25	0.130	0.033	4.810	0.213	12.127
50	0.217	0.047	14.113	0.187	18.703
75	0.353	0.073	26.213	0.267	24.837
100	0.497	0.120	43.040	0.267	31.613
200	0.950	0.197	128.847	0.360	70.710
250	1.153	0.267	215.013	0.437	84.610
500	2.530	0.607	1070.317	0.950	184.763
750	3.883	0.800	17950.257	1.787	233.550
1325	4.460	0.957	22288.270	1.550	252.433

TABLE VII
FEATURE GROUP IMPORTANCE.

FG	used	weighted
MV	66.2%	57.4%
Bias	5.1%	16.0%
RO	4.8%	7.1%
CP	5.4%	6.7%
EC	4.7%	4.5%
DTC	5.2%	4.4%
EE	8.7%	3.8%

TABLE VIII
PERFORMANCE OF THE PIPELINE ON THE *golub* DATA SET.

ϵ	AUC	F_1	n_{FAF}	n_{FAW}	NPR	WA	TFA
2.000	1.000	1.000	25	5	1	KNN	0
2.000	1.000	1.000	25	5	1	LR	0
2.000	1.000	1.000	25	5	1	RF	1
2.000	1.000	1.000	50	15	1	KNN	0
2.000	1.000	1.000	50	15	1	RF	1
1.395	0.844	0.551	200	15	3	LR	1
1.501	0.842	0.659	200	10	3	LR	1
1.384	0.802	0.582	200	5	10	LR	0
1.395	0.728	0.667	500	5	1	LR	0
1.327	0.721	0.607	500	5	5	LR	0

TABLE IX
PERFORMANCE OF THE PIPELINE ON THE *secom* DATA SET.

ϵ	AUC	F_1	n_{FAF}	n_{FAW}	NPR	WA	TFA
1.718	0.873	0.845	50	5	1	KNN	0
1.778	0.892	0.886	50	10	1	KNN	0
1.736	0.893	0.843	50	15	1	KNN	0
1.726	0.872	0.854	100	5	1	KNN	0
1.723	0.872	0.852	200	5	1	KNN	0
0.500	0.500	0.000	100	10	10	LR	0
0.499	0.499	0.000	200	5	10	KNN	0
0.483	0.483	0.000	500	10	5	KNN	0
0.445	0.445	0.000	500	5	3	LR	1
0.437	0.437	0.000	200	5	5	LR	1

E. Evaluation on publicly available data

For the *golub* data set, a multidimensional hyperplane exists, separating the two classes without classification errors. For certain parametrizations, the pipeline scores an averaged $AUC = 1$, which has also been achieved by Guyon et al. [20]. Table VIII shows the top and last five pipeline parametrizations, sorted by $\epsilon = AUC + F_1$. Keeping the original 833 features in mind, especially the first three rows represent a major benefit in terms of computational complexity because of an enormously reduced feature space. The optimum solution is in terms of *AUC* is possible with other parametrizations as well.

Generally, the *KNN* wrapper leads to higher performing models on the *golub* data set ($corr(KNN, AUC) = 0.271$). Also, a high *NPR* performs better ($corr(NPR, F_1) = 0.126$). Also beneficial is the TFA technique ($corr(TFA, AUC) = 0.138$). The LR wrapper performs badly on the *golub* data set ($corr(LR, AUC) = -0.272$) in contrast to the results based on the automotive data set.

The scores in terms of ϵ on the *secom* data set are promising as well (again, the top and last five results regarding ϵ are shown in Table IX) but lower compared to the $F_1 = 0.947$ achieved by Arif et al. [21] with a higher manual effort. The five highest ϵ scores have been achieved after reducing the 591 dimensional feature space to 25 and 50 features, respectively. The RF wrapper was included in this evaluation and scored $\epsilon_{RF,avg} = 0.912$ averaged over all parametrizations ($\sigma_{\epsilon,RF} = 0.241$). It is not displayed in the aforementioned table because KNN and LR yielded higher results ($\epsilon_{LR,avg} = 0.983$ and $\epsilon_{KNN,avg} = 0.961$) with a higher standard deviation ($\sigma_{\epsilon,LR} = 0.298$ and $\sigma_{\epsilon,KNN} = 0.356$).

A high *NPR* decreased the classification performance in this case ($corr(NPR, F_1) = -0.767$) – which is also outlined by Table IX, where the top 5 lines share low value of $NPR = 1$. The performance was affected negatively by the *TFA* technique ($corr(TFA, AUC) = -0.119$) and the RF wrapper algorithm ($corr(RF, AUC) = -0.157$).

VII. SUMMARY

The final scenario, including all averaged runtimes, is shown in Table X. First, the feature space was prepared and features with variance close to zero were filtered out. This already reduced the multi-thousand dimensional feature space to 1325 features after preparation (n_{FAP}). These were processed and ranked by the filter layer in $4.460s$ using all filter measures.

According to this ranking, the top ranked features after the filter layer (n_{FAF}) were passed to the wrapper. The wrapper training time increases when more features are being passed by the filter. The wrapper layer is usually computationally more expensive (see Figure 5) when processing the same amount of features. The model training time is given in the last row.

Together with n_{FAW} this spans up a matrix with AUC scores. For each column of n_{FAW} the corresponding model training time is given – which also increases when more features are processed.

To adapt the pipeline to other applications, the following recommendations may be helpful: Which wrapper to choose may be strongly dependent on the data set. In case of the *automotive* data set, the LR wrapper yielded a high AUC and F_1 while requiring a minimum of computational power. In contrast, LR yielded low results as wrapper for the *golub* dataset. Also, our wrapper evaluation may be biased towards the LR wrapper, since we used the same algorithm for modeling. Using the same algorithm as wrapper and for final modeling may be beneficial. In this case, combining wrapper and model layer might be possible.

TFA is a useful technique, if the number of features passed to the wrapper has to be very low (e.g. 5). In most cases, the available computing power allows to process 200 or more features. This featurespace will very likely be a superset of the featurespace selected by TFA.

After evaluating all filter measures introduced in Section V-B, we recommend using a subset of filter measures that is tailored to the needs of the application. If too many filter measures are used at the same time, the filter layer can be computationally more expensive than "cheap" wrapper layers, e.g., wrappers using LR. In case of an LR wrapper, the necessity of using a filter layer at all, must be evaluated on a case-by-case basis.

Averaging the pipeline performance in terms of AUC across all trained models, the highest performance was achieved with $n_{FAF} = 100$ and 200, respectively. Further increasing n_{FAF}

TABLE X
OVERVIEW OF PIPELINE PERFORMANCE AND RUNTIME USING AN RF WRAPPER AND AUC.

n_{FAP}	$filter_{all}$	n_{FAF}	wrapper training time [s]	5	10	15	25	50	500	n_{FAW}
		25	12.127	0.908	0.926	0.936	0.944	-	-	
		50	18.703	0.908	0.929	0.936	0.944	0.953	-	
1325	4.460s	100	31.631	0.907	0.929	0.938	0.944	0.953	-	
		200	70.710	0.906	0.929	0.937	0.945	0.953	-	
		500	184.763	0.901	0.926	0.934	0.943	0.952	0.951	
			model training time [s]	0.105	0.143	0.166	0.177	0.159	0.255	

did not yield higher results, but only leads to a higher wrapper training time.

There may be scenarios where the wrapper or filter layer is computationally more expensive than the model layer (such as in that case where we used an LR model). In case an increased number of features does not affect the generalization of the final model negatively, the wrapper and / or filter layer may be unnecessary. As described above using $n_{FAF} = 200$, $n_{FAW} = 50$ represents a good compromise between a high model performance, a low training time, and a low feature selection time. This results, e.g., in a total averaged time for the complete feature selection process t_{total}:

$$\begin{aligned} t_{total,avg,RF} &= t_{filter} + t_{wrapper} + t_{model} \\ &= 4.460s + 70.710s + 0.159s = 75.329s \end{aligned}$$

Or $0.957s + 0.360s + 0.159s = 1.476s$ using the proposed subset of filter measures $filter_{sub}$ and an LR wrapper.

Although the optimization of the pipeline took more than 2 weeks on a 32 core, 96GB workstation, the pipeline can be considered absolutely scalable once the hyperparameters are set. Even based on $t_{total,avg,RF}$, approximately 1100 models can be built in R per day on a single core machine.

VIII. CONCLUSION AND OUTLOOK

In this article we evaluated a variety of different algorithms and chained them in the most efficient way to create an automotive feature selection pipeline. The proposed pipeline involves multiple layers of different computational complexity, ensures maximum autonomy, and low computational cost. The pipeline already yielded promising models, awaiting their use to diagnose cars in the workshop.

Future work will examine the suitability of different machine learning models (e.g., neural networks, support vector machines, and random forests) for the given context. Furthermore, we will evaluate options how to increase throughput and scalability of the pipeline (e.g., using massive parallelization in a cluster) in order to be able to process data resulting from even more cars and engines.

ACKNOWLEDGMENT

This work was supported by the BMW AG and sheer driving pleasure.

REFERENCES

[1] BMW AG. (2015) Geschäftsbericht. Munich. [Online]. Available: https://goo.gl/z3cyM3 (August 2016)

[2] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*, 2nd ed. New York: Wiley, 2001.

[3] Y. Saeys, I. Inza, and P. Larranaga, "A review of feature selection techniques in bioinformatics," *Bioinformatics*, vol. 23, no. 19, pp. 2507–2517, 2007.

[4] R. Prytz, S. Nowaczyk, T. Rögnvaldsson, and S. Byttner, "Predicting the need for vehicle compressor repairs using maintenance records and logged vehicle data," *Engineering Applications of Artificial Intelligence*, vol. 41, pp. 139–150, 2015.

[5] Z. Tian, "An artificial neural network method for remaining useful life prediction of equipment subject to condition monitoring," *Journal of Intelligent Manufacturing*, pp. 227–237, 2012.

[6] R. Xu and D. Wunsch, "Survey of Clustering Algorithms," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, 2005.

[7] R. Ahmed, M. E. Sayed, S. A. Gadsden, J. Tjong, and S. Habibi, "Automotive Internal-Combustion-Engine Fault Detection and Classification Using Artificial Neural Network Techniques," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 1, pp. 21–33, 2015.

[8] J.-H. Thomas and B. Dubuisson, "A Diagnostic Method using Wavelets Networks: Appication to Engine Knock Detection," *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 1, pp. 244–249, 1996.

[9] A. Azarian and A. Siadat, "A global modular framework for automotive diagnosis," *Advanced Engineering Informatics*, no. 26, pp. 131–144, 2012.

[10] T. C. Müller, O. Krieger, A. Breuer, K. Lange, and T. Form, "A Heuristic Approach for Offboard-Diagnostics in Advanced Automotive Systems," *SAE World Congress*, vol. 2009-01-1027, pp. 344–351, 2009.

[11] T. Golub, D. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. Mesirov, H. Coller, M. Loh, J. Downing, and M. Caligiuri, "Molecular classification of cancer: class discovery and class prediction by gene expression monitoring," *Science*, vol. 286, no. 5439, pp. 531–537, 1999.

[12] M. Lichman. (2013) UCI Machine Learning Repository. [Online]. Available: http://archive.ics.uci.edu/ml/ (August 2016)

[13] I. Kononenko, "Estimating Attributes: Analysis and Extensions of RELIEF," *Lecture Notes in Computer Science*, vol. 784, pp. 171–182, 2005.

[14] V. Bolón-Canedo, N. Sánchez-Maronno, and A. Alonso-Betanzos, "A review of feature selection methods on synthetic data," *Knowledge and Information Systems*, vol. 34, no. 3, 2005.

[15] S. Breker, A. Claudi, and B. Sick, "Capacity of Low-Voltage Grids for Distributed Generation: Classification by Means of Stochastic Simulations," *IEEE Transactions on Power Systems*, vol. 30, no. 2, pp. 689–700, 2015.

[16] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A Library for Large Linear Classification," *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[17] J. Davis and M. Goadrich, "The relationship between Precision-Recall and ROC curves," *ICML '06 Proceedings of the 23rd international conference on Machine learning*, vol. 23, pp. 233–240, 2006.

[18] J. Huang and C. X. Ling, "Using AUC and Accuracy in Evaluating Learning Algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005.

[19] C. E. Metz, "Basic Principles of ROC Analysis," *Seminars in Nuclear Medicine*, vol. 8, no. 4, pp. 283–298, 1978.

[20] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene Selection for Cancer Classification using Support Vector Machines," vol. 46, no. 1, pp. 389–422, 2002.

[21] F. Arif, N. Suryana, and B. Hussin, "International Journal of Computer Applications," *International Journal of Computer Applications*, vol. 69, no. 22, pp. 35–40, 2013.