A Predictive Maintenance Methodology: Predicting the Time-to-Failure of Machines in Industry 4.0

Marwin Züfle*, Joachim Agne*, Johannes Grohmann*, Ibrahim Dörtoluk† and Samuel Kounev*
*Faculty of Mathematics and Computer Science, University of Würzburg, Germany
†ZF Friedrichshafen AG, Schweinfurt, Germany
Email: marwin.zuefle@uni-wuerzburg.de, samuel.kounev@uni-wuerzburg.de

***Abstract*—Predictive maintenance is an essential aspect of the concept of Industry 4.0. In contrast to previous maintenance strategies, which plan repairs based on periodic schedules or threshold values, predictive maintenance is normally based on estimating the time-to-failure of machines.Thus, predictive maintenance enables a more efficient and effective maintenance approach. Although much research has already been done on time-to-failure prediction, most existing works provide only specialized approaches for specific machines. In most cases, these are either rotary machines (i.e., bearings) or lithium-ion batteries. To bridge the gap to a more general time-to-failure prediction, we propose a generic end-to-end predictive maintenance methodology for the time-to-failure prediction of industrial machines. Our methodology exhibits a number of novel aspects including a universally applicable method for feature extraction based on different types of sensor data, well-known feature transformation and selection techniques, adjustable target class assignment based on fault records with three different labeling strategies, and the training of multiple state-of-the-art machine learning classification models including hyperparameter optimization. We evaluated our time-to-failure prediction methodology in a real-world case study consisting of monitoring data gathered over several years from a large industrial press. The results demonstrated the effectiveness of the proposed methodology for six different time-to-failure prediction windows, as well as for the downscaled binary prediction of impending failures. In this case study, the multi-class feed-forward neural network model achieved the overall best results.**

***Index Terms*—Predictive Maintenance, Time-to-Failure, Industry 4.0, Sensors, Feature Engineering, Machine Learning.**

I. Introduction

Rapid developments in the Internet of Things (IoT), continuous miniaturization of sensors, and increasing digitalization are leading to new applications in areas such as Industry 4.0 and Industrial IoT. Due to these developments, monitoring data of industrial machines can be easily stored, resulting in huge data sets. These can then be used to analyze the machines, with predictive maintenance (PdM) being a typical application.

Nowadays, many companies still follow a periodic (PM) or condition-based maintenance (CBM) approach. While in PM, industrial machines are maintained at regular intervals, CBM involves defining threshold values for particular sensors, which trigger maintenance operations when exceeded. PM often leads to a waste of personnel and material, since in many cases, maintenance is not necessary and could be postponed. A study by Mobley identified that this waste is responsible for about one third of all maintenance-related costs [1]. In contrast, CBM requires a high level of expert knowledge to define the threshold values. Furthermore, the analysis of threshold values is still a static, reactive approach, where unplanned downtimes still occur often. Emerson and the Wall Street Journal estimate the cost of these unplanned downtimes in manufacturing to approximately 50 billion USD per year [2], [3]. In contrast to PM and CBM, PdM aims at predicting deteriorating health by analyzing earlier monitoring data and learning from previous machine failures. A report by PricewaterhouseCoopers and Mainnovation revealed that PdM increases machine uptime by 9% and machine life by 20%. In addition, according to a report by the World Economic Forum [4], PdM reduces the costs of scheduled repairs by 12% and maintenance costs in general by 30%, while achieving 70% less breakdowns. Moreover, predicting upcoming machine failures also increases employee satisfaction, as technicians no longer have to be called to emergencies spontaneously and immediately, but can plan and prepare for the maintenance actions in advance.

Due to this high practical relevance of PdM for Industry 4.0, much research has already been done in this field. However, existing approaches are typically strongly tailored to particular use cases, so that the models can hardly be transferred to other applications. In contrast to these highly specialized solutions, we propose a generic end-to-end PdM methodology for time-to-failure prediction of industrial machines. The novel contribution compared to existing works is the end-to-end design, especially with respect to a universally applicable feature extraction based on integral values, which does not rely on profound expert knowledge. Furthermore, machine failures are predicted at different time horizons, while investigating the application of different class labeling strategies. We evaluate our proposed end-to-end methodology using real production data from a large-scale press. This type of machine has not been studied before and differs significantly from the types of machines commonly studied, namely bearings, lithium-ion batteries, and hard disk drives.

The rest of this paper is structured as follows: In Section II, we briefly describe related works in the domain of PdM and machine failure prediction. Then, we present our proposed PdM methodology for time-to-failure prediction in Section III. In Section IV, we evaluate our approach in the context of a real-world case study of a large-scale industrial press, followed by a discussion in Section V. Finally, we conclude the paper with a summary and an outlook on future work in Section VI.

II. Related Work

In general, predictive maintenance can be divided into two steps: (i) time-to-failure prediction and (ii) maintenance scheduling based on these time-to-failure predictions. In this paper, however, we focus on the first step, namely the time-to-failure prediction. Existing approaches in this area typically provide solutions designed for a specific machine. The most frequently analyzed machine type are rotary machines. Here, vibration data provide highly relevant indicators for machine health prediction. Zhang *et al.* proposed a time-to-failure prediction approach for bearings based on such vibration data using principal component analysis (PCA), hidden Markov model, and stochastic model for failure prediction [5]. In this case, Zhang *et al.* modeled the prediction problem as a regression task. Tobon-Mejia *et al.* introduced a similar approach, but they used wavelet decomposition instead of PCA and a mixture of Gaussian hidden Markov models [6].

Instead, Shuang *et al.* applied PCA in combination with a support vector machine (SVM) for binary classification of faulty bearings [7]. Huo *et al.* also employed binary classification using SVM, but used a variation of permutation entropy for feature extraction and Laplacian score for feature selection [8]. Unlike the other approaches, Amarnath *et al.* used sound signals instead of vibration data for binary classification of faulty bearings using a C4.5 decision tree [9].

Several authors have also applied multi-class classification to predict different bearing fault types. One such approach is by Zhang *et al.*, employing permutation entropy on vibration data, ensemble empirical mode decomposition (EEMD), and multi-class SVM [10]. Another approach based on EEMD was presented by Qin *et al.* [11]. However, they used random forest to classify fault types and compared EEMD with wavelet decomposition, concluding that EEMD provided better results. The use of empirical mode decomposition in combination with energy entropy and a feed-forward neural network was described by Ali *et al.* [12]. In contrast, Li *et al.* applied fast Fourier transform, root mean square maps, and convolutional neural networks for the classification of different bearing fault types [13]. However, all of these approaches used multi-class classification only to distinguish between different fault types and not to predict impending failures with different lead times.

The main shortcoming of the above approaches is that they are strongly tailored to vibration or sound data and therefore not transferable to domains where such data are not applicable or not as meaningful as for rotary machines. In addition, all of these approaches use a rigid prediction procedure without comparing the influence of different class labeling strategies.

III. Time-to-Failure Prediction Methodology for Industrial Machines

The time-to-failure methodology proposed in this paper is designed specifically for industrial machines. Therefore, the availability of sensor monitoring data is assumed. Section III-A presents the different categories of sensor data covered by the methodology. Next, Section III-B shows the feature selection included in the methodology along with two possible feature normalization techniques. Section III-C explains three different target labeling methods, followed by model learning in Section III-D. Finally, Section III-E describes the aggregation process for the predictions.

A. Extracting Features from Sensor Data

A sensor, in the context of industrial machines, is any digital measuring instrument that monitors either a particular component or the executed program of the machine. We distinguish sensors in four different categories, treating them in different ways: (i) physical units, (ii) paired units, (iii) temporal information, and (iv) program information.

Sensors in the *physical units* category are sensors that measure the current state or condition of a machine component, for example, the position of an axis, pressure, or temperature.

Sensors in the *paired units* category are similar, but they additionally provide a specified target value for the measured state or condition. The difference between the actual and target value can be used as a measure for the performance of the machine when running the implemented execution program.

In contrast, sensors in the *temporal information* category provide data that can be used to compute the duration of different events in the manufacturing process, typically the execution of certain manufacturing steps. Based on such data, a change in the health state of the machine can be detected. For example, in case of a leakage, the machine may take longer to build up a certain target pressure, which in turn may result in a longer duration.

Finally, sensors in the *program information* category provide data that can be used as an indicator to split the measurement data of an entire manufacturing process into multiple phases, for example, drilling, milling, tool change.

As part of the feature extraction process, our methodology first uses data from program information sensors to partition all sensor measurements into multiple separate parts representing different phases or activities in the manufacturing process.

Next, the first feature—the duration—is computed for each phase by subtracting the first timestamp from the last timestamp. For physical and paired units, a different feature extraction approach is used in order to gain insight into the performed execution. For most machine learning algorithms, it is not applicable to simply use the measured time series as input, since the durations of different executions vary, resulting in time series with different lengths. However, machine learning algorithms always require the same input feature dimension. Moreover, passing entire measurement time series as features leads to an explosion of the feature dimension, which slows down model learning tremendously, making it hardly applicable. Finally, the mere use of measurement time series can lead to misclassifications if the recording is slightly shifted in time, since the actually related features are then no longer considered as related by the machine learning model. Therefore, f or physical units, the integral of the measured values is computed; for paired units, the difference between the target values and the actual values is computed. This is illustrated in Figure 1, where a measured physical unit is

plotted against the measurement time and the integral for an exemplary phase in the manufacturing process is depicted in blue. The integral changes when either the intensity (i.e., the value range) of the physical unit or the duration of the phase increases or decreases, as indicated by the red arrows in Figure 1. Such changes typically indicate faulty machine behavior, since the execution should follow a strict sequence of instructions, although small variations are normal. The integral function provides a much more robust representation of these natural variations than the direct use of measurement data.

B. Feature Handling

The described feature extraction procedure, when applied for typical industrial machine scenarios with multiple physical unit sensors and multiple phases per manufacturing process, would result in a high number of extracted features. To avoid distortions during model learning and to keep the model building time as short as possible, only the most relevant features should be selected. For this purpose, the methodology includes a feature selection technique based on the Pearson correlation coefficient, which is computed between each feature and the target label. To reduce the dimensionality of the feature space, the operator can set a threshold value and specify whether only features with a correlation coefficient greater than the threshold value should be considered for model learning or also features with a correlation coefficient less than the threshold value. In the latter case, strongly positively and strongly negatively correlated features are included in the model learning.

However, since the values of different physical units are typically defined in different scales, the features must also be normalized before they can be used as input for machine learning models. For this purpose, our methodology offers two conventional techniques for feature normalization: min-max and Z-score normalization. The min-max normalization scales all values of a feature into the range of $[0, 1]$ by subtracting the minimum value from each value and dividing it by the difference between the maximum and minimum value. In contrast, the Z-score normalization transforms all values of a feature so that their new distribution has a mean value of 0 with a standard deviation of 1. To this end, it subtracts the mean value from each value and divides it by the standard deviation. Compared to the min-max normalization, the Z-score normalization is much more robust against outliers and does not require the absolute minimum and maximum values; however, it does not transform the values into a fixed range.

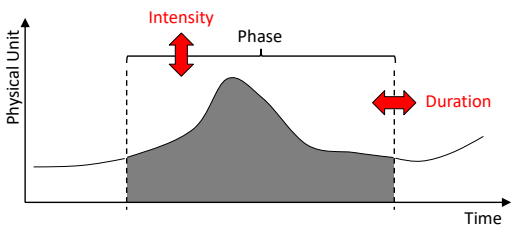

Fig. 1. Integral-based feature extraction.

C. Target Class Mapping

After the feature engineering process, the target labels must be created and mapped to the measurement data. To this end, our methodology assumes the availability of failure records with timestamps. The timestamps are matched to the timestamps in the sensor measurement data, computing the time-to-failure of each instance in the data set. Although the time-to-failure could directly be used as a label to learn regression models, the methodology discretizes the time-to-failure in terms of classes to learn classification models instead of regression models. This design decision is based on the fact that most real-life scenarios for predictive maintenance in Industry 4.0 do not require time-to-failure information at the granularity of seconds or minutes but rather of hours or days. This time horizon lends itself well to modeling using classes.

In order to derive the classification labels, our methodology applies three different labeling techniques. In each case, the operator must first specify a fixed number of prediction windows of interest, which are mapped to different time-to-failure classes. For instance, if the operator would specify the prediction windows of 12 hours and 1 day, the resulting discretized time-to-failure classes would represent failures in the three time horizons: $[0, 12)$, $[12, 24)$, and $[24, \infty)$ hours.

The first labeling technique is a *simple binary classification* per class. For each class, a vector of labels is created that is set to 1 if the time-to-failure of the particular instance is within the time-to-failure interval of the class and 0, otherwise. Only for the last class, no vector is created since this class is already covered by all other classes. If all other classes are 0, this implies a 1 for the last class. This labeling approach is illustrated in Figure 2 in the middle; it requires a separate prediction model for each class except for the last one.

The second labeling technique also models the classes separately, but contrary to the simple binary classification, the vector of class labels is set to 1 if the time-to-failure of the respective instance is smaller than the upper bound of the time-to-failure interval of the class. Thus, for a single instance, multiple classes can have a label of 1, while in the simple binary case, only the highest class would be set to 1. This labeling approach is illustrated at the bottom of Figure 2. Similar to the simple binary labeling technique, it results in a separate prediction model for each class except for the last one. Here, the last class would contain even less information, since it would be 1 for each instance. In the following, we refer to this labeling technique as *stacked binary*.

Finally, the third labeling technique is *multi-class labeling*, where a single vector of class labels is created containing the name of the time-to-failure interval class into which the time-to-failure of the corresponding instance falls. The top part of Figure 2 represents this labeling approach. Unlike the first two labeling techniques, this technique requires only a single prediction model.

D. Model Learning

To learn a model for time-to-failure prediction, the methodology uses four different methods. However, similar to the

implemented feature selection and transformation techniques, other algorithms would be possible as well and could be incorporated into the methodology. The four methods currently included in the time-to-failure prediction methodology are:

- Random Forest (RF): An ensemble method combining multiple decision trees using bagging [14],
- eXtreme Gradient Boosting (XGBoost): An ensemble method involving multiple decision trees using gradient boosting [15],
- Feed-Forward Neural Network (FFNN): A neural network architecture passing the information from input to output without cycles [16], and
- AutoML: A framework that automatically trains several models of different machine learning methods, optimizes their hyperparameters, and creates stacked ensembles from the best of these individual models [17].

For each of these methods, hyperparameter optimization is applied using the grid-search technique. That is, the operator can specify a list of possible configurations for each parameter. However, in the case study (see Table I), we also offer parameter configurations that have already achieved satisfactory results in practice and, therefore, can be adopted if no dedicated expert knowledge is available. For RF, the adjustable parameters are the number of decision trees *ntree* and the number of predictors *mtry* to choose from at each split. For XGBoost, the parameters to be defined are *eta*, *max_depth*, and *gamma*. As the number of configurable parameters of FFNN is very large, a predefined architecture is implemented consisting of five dense layers with an adjustable dropout after each layer. The rectified linear unit (ReLU) is used as an activation function. For the final output, the sixth layer applies the softmax activation function. Here, the number of nodes equals the number of classes defined by the operator. The design of this pre-defined FFNN architecture is based on Chollet and Allaire [18]. However, the resulting FFNN still provides a wide range of parameters, so the methodology splits the grid search into two iterations. The first grid search aims at estimating the required batch size and the overall complexity of the model by varying the parameter settings for the batch size, the number of nodes per layer, and the dropout. For this first iteration, the number of nodes and dropout are the same for all layers in order to keep the number of permutations manageable. Based on the top results of this first grid search, a second grid search is performed considering only the parameter settings that are included in these top results. Here, the number of nodes per layer and the dropout after each layer are set per layer. Thus, the pre-selected parameters are fine-tuned in this second iteration. To learn the FFNN models, the *adam* optimizer is used with categorical cross entropy as a loss function. Lastly, the methodology does not include hyperparameter optimization for AutoML, since AutoML already performs such an optimization internally.

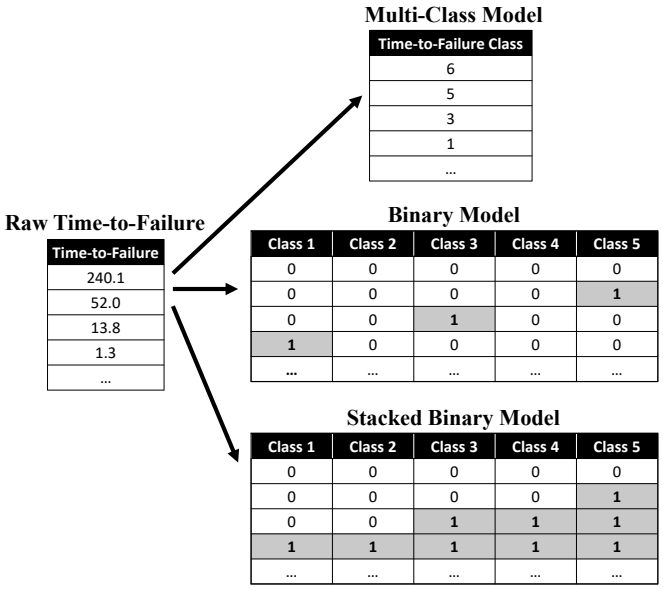

Fig. 2. Applied labeling techniques. For the binary models, the entries with 1 are marked grey.

E. Prediction Aggregation

Regardless of the analyzed labeling method, each prediction method returns only a single time-to-failure class when the models are applied. With respect to the multi-class model, the output is simply the predicted class. However, for the binary and stacked binary labeling methods, during the prediction process, several of the binary models may predict 1, namely an impending failure within the respective time window. In this case, only the time-to-failure class with the shortest lead time is returned as the final prediction.

IV. Case Study

This section presents a case study evaluating the proposed methodology for time-to-failure prediction by applying it to a large-scale industrial press.

A. Case Study Details

The case study analyzes data from a large-scale industrial press that was monitored over several years. During this period, every 100th stroke was recorded with a resolution of 1 millisecond. A stroke takes about four seconds during which the press processes eight different phases. As a data quality check, we sorted out recordings that did not consist of exactly these eight phases or had a processing time that was too short or too long. The phases are identified in the sensor recordings based on the recorded *command*, which can be seen as sensor data containing *program information*. Next, the timestamps of the sensor measurements are used as *temporal information* to compute the duration of each processing phase and of the entire stroke, resulting in 9 duration features. Besides the timestamps and the program command, 118 sensor measurements are available for each recording entry. This results in 118 physical unit sensors, 32 of which are paired units; that is, there are 16 pairs of target and actual values. For these pairs, the deviation is computed and used as physical unit

sensor data. In total, this results in $9 + (118 + 16) \times 8 = 1081$ features per stroke in a data set containing 47152 strokes.

To train the feed-forward neural network (FFNN), the features are transformed using the Z-score normalization. For the other model learning methods applied, both normalization techniques were tested, but the best predictions were obtained without any normalization. In addition, correlation-based feature selection was used to reduce the number of features. However, as the reduced feature set did not improve the prediction performance, in the following, we show the results of using all features for model learning.

The chosen class boundaries were 5 hours, 12 hours, 1 day, 2 days, and 1 week, resulting in the classes 1 to 6, with the time-to-failure falling into the range of $[0, 5)$, $[5, 12)$, $[12, 24)$, $[24, 48)$, $[48, 168)$, and $[168, \infty)$ hours, respectively. These limits were selected because predictions in these time windows allow technicians sufficient lead time to take countermeasures.

Table I shows the parameter settings for the applied grid-search-based hyperparameter tuning. Note that for classification, the default for *mtry* for random forest (RF) is set to $\lfloor\sqrt{\#\text{features}}\rfloor$, which yields 32. The other two settings for *mtry* are obtained by multiplying the square root of the number of features by factors of 2 and 4, respectively, followed by rounding. The XGBoost parameter specification is based on the recommendations of Jain [19]. Although AutoML performs hyperparameter optimization internally, one parameter, namely the maximum runtime, must be specified. For the binary and stacked binary models, we set the maximum runtime to 6 hours per binary class, resulting in a maximum total runtime of 30 hours. For the multi-class model, we set the maximum runtime to 22 hours. Although this may appear to be a bias, it does not affect the final prediction results since the maximum runtime was not reached anyway due to early stopping.

To assess the quality of the time-to-failure prediction methods, we compute accuracy, recall, precision, F1-score, and Cohen's kappa coefficient [20] using a 5-fold cross-validation [21]. While the first four measures are very commonly used in machine learning-based classification tasks, Cohen's kappa coefficient is less frequently observed. Therefore, we briefly describe it here. Cohen's kappa coefficient measures the quality of a predictor by comparing the predictions to a random choice predictor based on class frequencies.

TABLE I
PARAMETER SETTINGS OF THE PREDICTION METHODS FOR THE GRID-SEARCH-BASED HYPERPARAMETER OPTIMIZATION

Method	Parameter	Settings
RF	*ntree*	500, 1000, 2000
	mtry	32, 65, 131
XGBoost	*eta*	0.05, 0.10, 0.30
	max_depth	6, 8, 10, 15
	gamma	0, 2, 10
FFNN	batch size	100, 500, 1000
	number of nodes	64, 128, 256, 512
	dropout	0.0, 0.2, 0.4

Formally, Cohen's kappa coefficient is defined by

$$\text{Kappa} = \frac{p_o - p_e}{1 - p_e}, \tag{1}$$

where p_o is the accuracy of the learned predictor and p_e is the expected accuracy of a random choice predictor that uses the class distribution of the training set as sampling frequencies. Cohen's kappa coefficient is more expressive than ordinary accuracy for imbalanced data sets. The distribution of classes in the data set used in this case study is illustrated in Table II. It can be clearly seen that the classes are highly imbalanced; for instance, the time-to-failure class 6 (i.e., no failure within the next week) occurs 1118 times in the data set, whereas all other classes are in the range of 7900 to 11000 instances.

TABLE II
THE DISTRIBUTION OF TIME-TO-FAILURE CLASSES.

Class	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
Count	10977	7978	8762	7906	10411	1118

B. Macro Results

In this section, the best models for the three different labeling techniques are compared over all classes. The measures recall, precision, and F1-score are calculated for each class and then combined using arithmetic mean, resulting in macro recall, macro precision, and macro F1-score. The class-wise results are presented in Section IV-C.

Note that for the binary or stacked binary labeling, the best model may actually consist of five different prediction methods and hyperparameter settings corresponding to the classes 1 to 5 (cf. Section III-C), while for the multi-class labeling, the best model comprises only a single prediction method with its best hyperparameter configuration. Table III displays the achieved accuracy, kappa, macro F1-score, macro recall, and macro precision for the best models of each combination of labeling and prediction method. Moreover, for the binary and stacked binary labeling methods, also the best results using a combination of different prediction methods are shown.

For the binary models, it can be observed that random forest yielded by far the worst result with respect to accuracy, kappa, macro F1-score, and macro recall. However, it achieved the second highest macro precision. XGBoost and FFNN performed rather similarly for all measures except macro precision, where XGBoost provided the best result and FFNN the worst. Finally, AutoML showed the overall best prediction when using binary labeling. For all measures except macro precision, it significantly outperformed the other prediction methods. For this reason, the overall best combination of binary models includes only AutoML models, which were all based on boosting, i.e., mainly XGBoost models and a few gradient boosting machine models. Since the best combination is equal to the application of the five best binary models of AutoML, both rows in Table III show the same values.

In terms of the stacked binary models, the first finding is that the prediction quality of all methods increased greatly. Here,

TABLE III

THE ACHIEVED ACCURACY, KAPPA, MACRO F1-SCORE, MACRO RECALL, AND MACRO PRECISION FOR THE BEST PARAMETER SETTINGS OF EACH INDIVIDUAL PREDICTION METHOD AS WELL AS THEIR OVERALL BEST COMBINATION FOR EACH LABELING APPROACH. THE BEST VALUES PER LABELING METHOD ARE HIGHLIGHTED IN BOLD, WHILE THE OVERALL BEST VALUES ARE HIGHLIGHTED IN BOLD AND WRITTEN IN ITALIC.

Labeling Method	Prediction Method	Accuracy	Kappa	Macro F1-Score	Macro Recall	Macro Precision
Binary	Best Random Forest	0.5670	0.5042	0.5796	0.6140	0.8065
	Best XGBoost	0.7183	0.6672	0.6933	0.7476	**0.8070**
	Best FFNN	0.7129	0.6519	0.6657	0.7320	0.6931
	Best AutoML	**0.7934**	**0.7475**	**0.7372**	**0.8036**	0.7555
	Best Combination	**0.7934**	**0.7475**	**0.7372**	**0.8036**	0.7555
Stacked Binary	Best Random Forest	0.7576	0.7009	0.7749	0.7698	0.8053
	Best XGBoost	0.8378	0.7995	0.8526	**0.8522**	**0.8621**
	Best FFNN	0.7568	0.6983	0.7703	0.7610	0.7835
	Best AutoML	0.8280	0.7863	0.8417	0.8371	0.8472
	Best Combination	**0.8409**	**0.8023**	**0.8533**	0.8492	0.8590
Multi	Best Random Forest	0.8763	0.8462	0.8797	0.8770	0.8845
	Best XGBoost	0.8862	0.8585	0.8920	0.8907	0.8946
	Best FFNN	***0.8972***	***0.8724***	***0.9000***	***0.9041***	***0.8951***
	Best AutoML	0.8408	0.8021	0.8519	0.8482	0.8571

random forest and FFNN provided similar results regarding all measures, with the achieved values of random forest being slightly higher. In addition, XGBoost outperformed AutoML with respect to all measures, although all models learned from AutoML were still based on boosting methods only. However, the best combination of models was achieved by a mixture of XGBoost and AutoML models. The best combination model used XGBoost with *eta* = 0.05, *max_depth* = 15, and *gamma* = 0 for classes 2 and 3, while AutoML models were used for classes 1, 4, and 5.

In contrast to this dominance of AutoML and XGBoost for the binary and stacked binary models, FFNN significantly outperformed AutoML in the case of multi-class labeling, despite the fact that AutoML learned a complex ensemble model consisting of eight gradient boosting machine models, seven XGBoost models, one linear model, two random forest models, and seven deep neural networks. Yet, XGBoost provided the second best results on all evaluation measures. Moreover, even random forest outperformed AutoML significantly. The results obtained by the multi-class FFNN were the highest across all labeling and prediction methods with an accuracy, kappa, macro F1-score, macro recall, and macro precision of 89.72%, 87.25%, 90.00%, 90.41%, and 89.51%, respectively. The configuration of this optimized FFNN was as follows:

- batch size = 500
- number of nodes per layer = $[256, 256, 512, 512, 256]$
- dropout per layer = $[0.2, 0.2, 0.0, 0.0, 0.2]$

Finally, when comparing the kappa values of the best prediction methods per labeling strategy, only the multi-class labeling approach achieved a value greater than 0.81, which according to Landis and Koch [22] is considered the lower limit of a near-perfect agreement. In numerical terms, the best multi-class labeling model significantly exceeded this limit, showing a kappa value of 0.8724, while the best stacked binary labeling model almost reached the limit with a kappa value of 0.8023, and the best binary labeling model remained far from it with a kappa value of 0.7545. These results could be explained by relationships between classes. When all six classes are modeled with a single prediction model, slight differences between the classes can be learned that are missed when analyzing each class individually. Therefore, the next section presents the prediction results by class.

C. Results by Class

In this section, the achieved prediction quality per class is evaluated in detail. Table IV shows the F1-score (F), recall (R), and precision (P) for the binary (B), stacked binary (S), and multi-class (M) labeling for each of the six time-to-failure classes. The best values per measure and class are highlighted in bold, while the worst values are underlined. Regarding the F1-score, it is apparent that the multi-class labeling performed best; it achieved the highest value for five of the six classes. Only for the last class (i.e., no machine failure within the next week), the stacked binary labeling approach performed better by 0.0081 percentage points. The binary labeling approach produced the worst F1-scores for all classes. Nevertheless, it achieved the same F1-score for class 1 as the stacked binary labeling approach. This is due to the fact that the

TABLE IV

THE ACHIEVED F1-SCORE (F), RECALL (R), AND PRECISION (P) PER CLASS FOR THE BEST MODELS OF EACH LABELING METHOD (LM), I.E., BINARY (B), STACKED BINARY (S), AND MULTI (M). THE BEST VALUES ARE WRITTEN BOLD, WHILE THE WORST VALUES ARE UNDERLINED.

	LM	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
F	B	0.8155	0.6935	0.7955	0.8595	0.9356	0.3234
	S	0.8155	0.7273	0.8142	0.8765	0.9403	**0.9460**
	M	**0.8807**	**0.8353**	**0.8772**	**0.9111**	**0.9589**	0.9379
R	B	0.8356	0.6161	0.7319	0.8129	0.9085	0.9168
	S	0.8356	0.6980	**0.8538**	0.8679	0.9158	0.9240
	M	**0.8913**	**0.8420**	0.8495	**0.9161**	**0.9587**	**0.9669**
P	B	0.7964	0.7933	0.8712	**0.9116**	0.9643	0.1964
	S	0.7964	0.7591	0.7781	0.8853	**0.9662**	**0.9690**
	M	**0.8703**	**0.8206**	**0.9068**	0.9062	0.9561	0.9105

labeling procedure for the first class is the same for both labeling strategies and both use the same AutoML model for class 1. This also applies to recall and precision. Moreover, for classes 1 to 5, the obtained F1-score of the binary labeling is close to the F1-score of the stacked binary labeling, it drops significantly for class 6. While the stacked binary labeling and the multi-class labeling achieved an F1-score of 0.9460 and 0.9379, respectively, the binary labeling only reached an F1-score of 0.3234 for class 6.

When analyzing recall, again the multi-class labeling approach performed best for five of the six classes. Only for class 3, stacked binary labeling outperformed multi-class labeling. Similar to the F1-score results, the difference is quite small with only 0.0043 percentage points. Again, the binary labeling performed worst for all six classes, while achieving the same results as the stacked binary labeling approach for class 1. Nevertheless, the binary labeling did not provide as significantly poor results as for the F1-score of class 6. The largest difference between the binary labeling and the second best labeling method is for class 3, where the binary labeling yielded a recall of 0.7319 and the multi-class labeling a recall of 0.8495. Since recall is the number of instances correctly predicted as positive relative to the total number of instances that are actually positive, these results suggest that all three labeling methods predict imminent machine failures quite well while keeping the number of missed imminent failures low.

Lastly, the multi-class labeling approach achieved the highest precision for three time-to-failure classes (i.e., classes 1 to 3), while the stacked binary labeling approach performed best for two classes (i.e., classes 5 and 6) and the binary labeling approach provided the best precision for class 4. In contrast, the stacked binary labeling performed worst for four classes, namely classes 1 to 4, while binary labeling yielded the same precision for class 1. Since precision is the number of instances correctly predicted as positive relative to the total number of instances predicted as positive, this implies that the stacked binary labeling approach provided comparatively many false alarms for classes with a short time-to-failure, while it delivered the least number of false alarms for the two classes with the highest time-to-failure. Similarly, the binary labeling approach resulted in the fewest false alarms for the classes indicating failures within the next 24 to 48 hours. However, the precision of the binary labeling for class 6 is by far the worst with only 0.1964, resulting in the low F1-score of the binary labeling. That is, the binary labeling approach predicted many instances as class 6, although the time-to-failure is actually much shorter. This demonstrates the need to compare different labeling strategies, as their performance varies greatly both between classes and in aggregate. Finally, the multi-class labeling provided least false alarms for the most urgent time-to-failure classes.

D. Details on the Best Predictions

This section provides more detailed insights into the overall best predictions, which were achieved by the feed-forward neural network using multi-class labeling. Table V reports the accumulated confusion matrix across all five folds, such that each instance in the data set is used exactly once in the test set. The columns of the confusion matrix show the actually observed (Ob) time-to-failure classes, while the rows represent the predicted (Pr) classes. The value in each cell shows the number of instances predicted for that particular set of observed and predicted class labels. The cells highlighted in green indicate the correctly predicted instances. The second cell from the left in the third row, for example, shows 398 instances that are predicted as "a failure will occur within the next 12 to 24 hours", while the failure actually occurred in the time window of 5 to 12 hours after the measurement.

The confusion matrix clearly shows that most instances are correctly predicted. Especially for higher time-to-failure classes, only few mispredictions are made, whereas most mispredictions occur for shorter time-to-failure classes, where the time difference between classes is also smaller. Moreover, the majority of the mispredictions are located directly in adjacent classes. Thus, the time-to-failure mispredictions mostly fall in the closest time-to-failure windows, which leads to only minor discrepancies for practical applications. Only relatively few failure instances exhibit higher prediction error (i.e., several cells apart in the confusion matrix).

Furthermore, for many practical applications, such a fine-grained time-to-failure prediction might not be required and instead a binary prediction with a prediction horizon of two days might be sufficient. To evaluate our approach for such scenarios, we reduced the confusion matrix with six classes to the binary prediction of whether the machine failure will occur within the next two days or not. That is, the classes 1 to 4 represent the "failure class", while the classes 5 and 6 are grouped as the "no failure" class. For this binary task, our predictive maintenance methodology predicts the failure with even (substantially) higher accuracy, F1-score, and kappa of 97.79%, 98.54%, and 94.03%, respectively.

V. Discussion and Threats to Validity

Although it is common practice to apply multiple binary prediction models to predict impending failures with different lead times (e.g., [23]), the results of our case study demonstrated that such an approach does not necessarily yield good results. In fact, the binary models actually performed the worst in our case study. However, this finding only represents this case study, while the ranking of the best labeling methods and machine learning algorithms could be completely different for

TABLE V
THE ACCUMULATED CONFUSION MATRIX OVER ALL FIVE FOLDS FOR THE FEED-FORWARD NEURAL NETWORK WITH MULTI-CLASS LABELING.

Pr \ Ob	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
Class 1	9719	755	243	131	144	20
Class 2	654	6635	462	86	78	12
Class 3	262	398	7671	235	42	18
Class 4	132	109	297	7283	166	11
Class 5	190	69	74	163	9959	18
Class 6	20	12	15	8	22	1039

another use case. Nevertheless, the results illustrate the need to test different class labeling strategies, as their influence can be even larger than the choice of machine learning method, as it was the case in our case study. Given that only integral and duration features were used, the methodology does not require in-depth expert knowledge, making it easily transferable to other application scenarios, even though it was evaluated here in only one concrete use case. Furthermore, the case study considered failures of a large industrial press, which have hardly been analyzed in the literature so far.

While the application runtime of the prediction models is negligible, the training and hyperparameter tuning takes a considerable amount of time. In practical application, the models should nevertheless be re-trained regularly to counteract possible concept drifts. However, due to the high computational effort, this re-training should be carried out off-line.

With respect to the prediction aggregation for the stacked binary model, we also compared the use of a strategy that analyzes the consistency of 1s within the predictions. However, using the time-to-failure class with the shortest lead time yielded the best results. Yet, there exist many other ways to aggregate the predictions of the binary models, such as incorporating class probabilities. As future work, we plan to develop different prediction aggregating methods and investigate their effects on the overall quality of the predictions.

We also plan to incorporate oversampling strategies to balance the number of instances of each class, as this has been shown to be beneficial in previous work [24]. Furthermore, we plan to extend the prediction methods to include recurrent neural networks. In addition, we envision combining the models of the different labeling methods into an overall model, as results suggest that different labeling methods perform better for different time-to-failure windows. This would allow us to create an ensemble that weights the prediction of each model with respect to the time-to-failure window. Finally, we plan to use our time series forecasting method Telescope [25] to forecast sensor values, providing additional information for the machine learning method.

VI. Conclusion

In this paper, we introduced a universal end-to-end predictive maintenance methodology for time-to-failure prediction of industrial machines. Unlike other time-to-failure prediction approaches that are specifically designed for certain machines, the proposed methodology includes a universal sensor handling and feature extraction approach based on integral values. Moreover, it also includes feature transformation, feature selection, adjustable target class labeling, and training of different machine learning models as well as their hyperparameter tuning. Thus, the proposed methodology provides an end-to-end approach from sensor input to time-to-failure prediction. However, the planning of targeted maintenance using this time-to-failure prediction was beyond the scope of this thesis. We evaluated the predictive maintenance methodology by applying it to a real-world case study predicting the time-to-failure of a large-scale industrial press with six different time windows. The results showed that a feed-forward neural network with multi-class labeling managed to achieve the best prediction quality in terms of accuracy, F1-score, and kappa.

References

[1] R. K. Mobley, *An Introduction to Predictive Maintenance*. Elsevier, Oct 2002.

[2] "How Manufacturers can Achieve Top Quartile Performance," https://partners.wsj.com/emerson/unlocking-performance/how-manufacturers-can-achieve-top-quartile-performance/, accessed: 2020-12-01.

[3] "Emerson Tackles Costly Downtime Losses in Industrial Process Facilities, Launches Reliability Management Consulting Service," https://www.emerson.com/en-us/news/corporate/reliability-consulting, accessed: 2020-12-01.

[4] World Economic Forum, "Industrial Internet of Things: Unleashing the Potential of Connected Products and Services," Tech. Rep., Jan 2015.

[5] X. Zhang, R. Xu *et al.*, "An integrated approach to bearing fault diagnostics and prognostics," in *Proceedings of the American Control Conf.* IEEE, 2005.

[6] D. A. Tobon-Mejia, K. Medjaher *et al.*, "A data-driven failure prognostics method based on mixture of gaussians hidden markov models," *IEEE Transactions on Reliability*, vol. 61, no. 2, pp. 491–503, 2012.

[7] L. Shuang and L. Meng, "Bearing fault diagnosis based on pca and svm," in *2007 Int. Conf. on Mechatronics and Automation*. IEEE, 2007.

[8] Z. Huo, Y. Zhang *et al.*, "A new bearing fault diagnosis method based on fine-to-coarse multiscale permutation entropy, laplacian score and svm," *IEEE Access*, vol. 7, pp. 17 050–17 066, 2019.

[9] M. Amarnath, V. Sugumaran, and H. Kumar, "Exploiting sound signals for fault diagnosis of bearings using decision tree," *Measurement*, vol. 46, no. 3, pp. 1250–1256, 2013.

[10] X. Zhang, Y. Liang *et al.*, "A novel bearing fault diagnosis model integrated permutation entropy, ensemble empirical mode decomposition and optimized svm," *Measurement*, vol. 69, pp. 164–179, 2015.

[11] X. Qin, Q. Li *et al.*, "The fault diagnosis of rolling bearing based on ensemble empirical mode decomposition and random forest," *Shock and Vibration*, vol. 2017, 2017.

[12] J. B. Ali, N. Fnaiech *et al.*, "Application of empirical mode decomposition and artificial neural network for automatic bearing fault diagnosis based on vibration signals," *Applied Acoustics*, vol. 89, pp. 16–27, 2015.

[13] S. Li, G. Liu *et al.*, "An ensemble deep convolutional neural network model with improved ds evidence fusion for bearing fault diagnosis," *Sensors*, vol. 17, no. 8, p. 1729, 2017.

[14] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[15] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2016.

[16] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[17] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[18] F. Chollet and J. J. Allaire, *Deep Learning mit R und Keras: Das Praxis-Handbuch von den Entwicklern von Keras und RStudio*. MIT Press, 2018.

[19] A. Jain, "Complete guide to parameter tuning in xgboost (with codes in python)," 2016.

[20] M. L. McHugh, "Interrater reliability: the kappa statistic," *Biochemia medica*, vol. 22, no. 3, pp. 276–282, 2012.

[21] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[22] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *biometrics*, pp. 159–174, 1977.

[23] J. Li, X. Ji *et al.*, "Hard drive failure prediction using classification and regression trees," in *44th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks*. IEEE, 2014.

[24] M. Züfle, C. Krupitzer *et al.*, "To fail or not to fail: Predicting hard disk drive failure time windows," in *Int. Conf. on Measurement, Modelling and Evaluation of Computing Systems*. Springer, 2020.

[25] A. Bauer, M. Züfle *et al.*, "Telescope: An automatic feature extraction and transformation approach for time series forecasting on a level-playing field," in *36th Int. Conf. on Data Engineering*. IEEE, 2020.