applied sciences

MDPI

Article

Stacked Ensemble of Recurrent Neural Networks for Predicting Turbocharger Remaining Useful Life

Peyman Sheikholharam Mashhadi *, Sławomir Nowaczyk and Sepideh Pashami

Center for Applied Intelligent Systems Research (CAISR), Halmstad University, 30118 Halmstad, Sweden; slawomir.nowaczyk@hh.se (S.N.); sepideh.pashami@hh.se (S.P.)

* Correspondence: peyman.mashhadi@hh.se

Received: 21 November 2019; Accepted: 15 December 2019; Published: 20 December 2019

Abstract: Predictive Maintenance (PM) is a proactive maintenance strategy that tries to minimize a system's downtime by predicting failures before they happen. It uses data from sensors to measure the component's state of health and make forecasts about its future degradation. However, existing PM methods typically focus on individual measurements. While it is natural to assume that a history of measurements carries more information than a single one. This paper aims at incorporating such information into PM models. In practice, especially in the automotive domain, diagnostic models have low performance, due to a large amount of noise in the data and limited sensing capability. To address this issue, this paper proposes to use a specific type of ensemble learning known as Stacked Ensemble. The idea is to aggregate predictions of multiple models—consisting of Long Short-Term Memory (LSTM) and Convolutional-LSTM—via a meta model, in order to boost performance. Stacked Ensemble model performs well when its base models are as diverse as possible. To this end, each such model is trained using a specific combination of the following three aspects: feature subsets, past dependency horizon, and model architectures. Experimental results demonstrate benefits of the proposed approach on a case study of heavy-duty truck turbochargers.

Keywords: predictive maintenance; remaining useful life; recurrent neural networks; LSTM; Stacked Ensemble

1. Introduction

Unplanned downtime, which means any period of time a machine is not available for use, has been shown to be one of the largest sources of cost and lost production time. Predictive Maintenance (PM), also known as conditioned-based maintenance, aims to mitigate these issues. As the name suggests, the promise of predictive maintenance is to predict upcoming failures, based on the actual condition (or state) of the system, in order to perform required maintenance at the best possible time. Doing so will increase system functionality and availability while reducing the collateral failure costs associated with corrective maintenance and redundant replacements caused by preventive maintenance.

There is a rich history of research on vehicle predictive maintenance focusing on single vehicle readouts [1,2]. However, relying on single readouts to predict possible future failures means conditioning the future on current observed state of the vehicle only. This corresponds to assuming that the future is independent of the past, given the current observable state, i.e., Markovian. In the context of automotive industry this assumption, however, is quite unrealistic due to limited number of sensors and high amount of noise in the data. More intuitively, the signs of future failures are not necessarily fully encoded in the current sensor readings.

There have been several methods developed over the years for predictive maintenance based on data sequences. They generally represent three categories—Autoregressive Moving Average (ARMA)-based, Hidden Markov Model (HMM)-based, and Recurrent Neural Networks (RNN)-based approaches.

The first category, ARMA-based, aims to predict the future values of a time series by regression over the past values of the dependent variable (Autoregressive Model), and also modeling the stochastic part of time series (Moving Average Model). As an example of the first category, Zhao et al. [3] adopted an 8th-order ARMA model for forecasting failures of semiconductor Assembly and Test (ATM) factory. They transformed the raw data to a Machine Factor indicator which is defined as a function of downtime and productive time, then the ARMA model is used on this indicator to predict future failures. More recently, Baptista et al. [4] proposed a pipeline consisting of combination of data-driven techniques and ARMA model to predict failures of engine valves.

Another alternative for sequence analysis is HMMs. They were considered suitable for predictive maintenance, since they can represent state of health of a component as hidden variables and the measurements as observed variables, while modeling the sequence through transition probabilities. Zhang et al. [5] proposed an integrated framework using HMM for Bearing Fault Diagnostics. First, they applied Principle Component Analysis (PCA) for feature extraction, then HMM is used to produce a health degradation signal which is fed as input to an adaptive prognostic method. Kamlu and Laxmi [6] used a combination of HMM and fuzzy model for maintenance of bus transportation systems. They generated several fuzzy rules that determine whether maintenance is required or not, where probabilities obtained from HMM are used as input to the fuzzy model.

This paper focuses on the third group of models, namely Recurrent Neural Networks (RNN). RNNs, due to their ability to model long term non-linear dependencies over time, were shown to be a promising approach in many domains, including predictive maintenance. Wang et al. [7] employed RNN for the case study of fault diagnosis of large-scale power plant. They proposed a meta-heuristic optimization algorithm for simultaneous structural design and parameter optimization of the RNN. Chen et al. [8] used RNN for fault prediction of motor bearing data. To feed the RNN with stationary signal, they tackle the non-stationary nature of the data by decomposing it using empirical model decomposition. The change in trend is obtained through entropy calculation and used as an input by RNN. Chen et al. [9] adopted Long Short-Term Memory (LSTM) networks for failure predictions of air compressors of heavy duty Volvo trucks. They concluded that using LSTMs leads to more consistency in predictions over time in comparison to models that ignore history, such as Random Forest. Kovalev et al. [10] made experiments with different sequence-based methods, including RNN, for predictive maintenance of housing and utility infrastructure. One of their main conclusions is that HMM works better for fault detection and classification while RNN models tend to be more useful for Remaining Useful Time (RUL) prediction. Vita et al. Bruneo and De Vita [11] investigated the advantages of LSTM in comparison to other learning models such as SVM and non-recurrent deep networks on the NASA jet engines dataset.

Despite all these developments related to predictive maintenance, in several practical settings the results still need improvements. In most scenarios these methods suffer from too many false positives while missing many failures due to challenges with low quality of data, limited availability of sensors, and high data imbalance.

In other domains where Machine Learning has faced similar challenges, turning to ensemble learning has been quite successful. Ensemble learning refers to a class of machine learning techniques in which the decisions from multiple models are combined to increase the overall performance in terms of both stability and accuracy. In order for the ensemble learning model to perform properly, diversity of the models is a key factor. Having a diverse set of models ideally means each model is more specialized towards specific parts of feature space, and consequently combining them will result in overall better modelling of all parts of feature space. For instance, Bootstrap Aggregation (Bagging) which is a well-known ensemble technique create diversity by multiple sampling with replacement (bootstrap sampling) from data and training separate model for each sample. After training all models, their predictions are aggregated to give the final aggregated predictions.

As another example of ensemble learning, "Stacked Ensemble" is a more advanced form of ensemble learning. Instead of calculating simple statistics of models predictions, it uses another model (meta model) on top of previous models predictions. Using a meta model leads to a more accurate combination of base models, and having diversity in base models makes finding this optimal combination more feasible.

However, in predictive maintenance the use of ensembles has been quite limited. Baraldi et al. [12] developed an ensemble of Kalman filters for predicting RUL of turbine blades. They used bagging to make the base models diverse and predictions of different models are combined through a weighted averaging where the weights are inversely proportional to the performance of each base model. Seera et al. [13] proposed an ensemble method consisting of Classification and Regression Trees (CART) using a modified Gini index based on the classification boundaries defined by fuzzy min-max neural networks for the case study of prognostics of induction motors. This modified CART is framed in a bagging ensemble framework similar to Random Forest. As far as we know, nobody used stacked ensembles for predictive maintenance.

The main contribution of this paper is combining the history-awareness of RNN models with stacked ensemble approach. To make the base models of the ensemble as diverse as possible, each base model is trained on a specific combination of three different aspects—feature subsets, past dependency horizon, and model architectures. By training models on different feature subsets we make different models concentrate on different parts of the feature space. Training on different past dependency horizons makes the models explore history of the vehicles at multiple resolutions. Lastly, the promise of using different modeling architectures is that they establish varying mappings between measurements and failures.

The remainder of the paper is structured as follows. In Section 2, data representation and the labeling mechanism for the task of vehicle predictive maintenance is described; Section 3 discusses feature engineering and, in particular, the covariate shift problem. Stacked ensemble mechanism is presented in detail in Section 4. Emphasising on an important requirement of ensemble learning, diversity, Section 5 defines different aspects to maintain diversity of base models. Section 6 presents the chosen model architectures, which is followed by experimental evaluation in Section 7. Finally, Section 8 concludes the paper.

2. Data Presentation

2.1. Dataset Decriptions

In this work we focus on vehicles that have encountered turbocharger failures during their operations—the goal is predicting the time to failure. Therefore, we limit the investigation only to this subset of the complete fleet.

As with most of the vehicles predictive maintenance tasks, the data comes from two separate sources [14]. One dataset contains vehicle on-board sensor information which is timestamped and indexed by vehicle ids. This dataset, which in our case is called Logged Vehicle Data (LVD), includes sensors measurements collected bi-weekly. There are more than 400 features, 83% of them numerical and the rest categorical. Numerical features represent trucks' usage-related measurements and categorical features represent trucks' configurations. Examples of usage features are "engine time", "fuel consumption", "time in top gear", "cruise driving distance", etc. Examples of configurations features are "engine type", "front axle type", "cabin version", etc.

Another dataset is the repairs dataset, which in our case study was limited to turbocharger replacement information. In this dataset at each row, the vehicle id and repair date are registered. If a vehicle gets repaired more than once, there will be more that one record with the same vehicle id.

One difficulty with this dataset is that the failures are very rare. Therefore, if we were to assign binary labels to LVD readouts by aligning the two datasets, only a very small portion of readouts would get labels. Hence, as a classification problem, it would be a very imbalanced dataset.

2.2. How to Label the Data?

One important challenge is to determine how to label the LVD readouts. There are two main approaches for tackling this problem: the sliding-box and time to next event (TTE) approaches. The former corresponds to formulating the problem as a classification task, while the latter as a regression task.

In a sliding-box model [1,15] the idea is to define whether a failure is happened within a predefined time interval from the time step t. More formally, each time step is assigned a binary label according to the following formula.

$$L_t = \begin{cases} 1 & \text{if there is failure in [t,t+}\tau\text{)} \\ 0 & \text{if there is no failure in [t,t+}\tau\text{)} \end{cases}$$

There are two main issues with this approach. The first issue is the need for tuning the time interval (τ) size, which can be difficult to choose.

Choosing a small interval makes the data highly imbalanced, since failures are rare in comparison to non-failures. It also means having very little time to react to a warning. On the other hand, large interval means that predictions far into the future, possibly before any failure symptoms develop, are as important as shorter ones. The second issue, more serious, is the sudden jumps from zero to one (non-failure to failure) at an arbitrary amount of time ahead of failure. In most cases the degradation of components is gradual, depending on in-operation time of the vehicle. By making training signal with such crisp boundaries we confuse the model by requiring this gradual degradation to match sudden jumps in the labels. In other words, the difference between two consecutive logged measurements does not necessarily correspond to the difference in their labels.

On the bright side, using sliding-box approach, one can directly decide about the maintenance action of a component before its failure since it directly solves for the decision making problem (failure or non-failure).

The second approach to labeling the data is Time to Event (TTE) or Remaining Useful Life (RUL) approach [16,17]. The idea is that instead of assigning binary labels, at each point in time, we assign time to the next event as the label [18,19]. The main drawback of TTE labeling approach is that there is no clear way to label machines for which no failures were reported. It is not an issue in our setting, where we only focus on the subset of the fleet with failures. Also, since it does not directly solve for the decision making task (failure, non-failure), another layer will be needed to convert TTE predictions to decisions.

On the other hand, TTE labeling has three main advantages. First, there is no need for tuning hyper-parameters such as interval size in the sliding-box method. The second advantage is that it eases the problem of imbalanced data which one can encounter in classification, allowing the model to use more informative training signal. The third advantage is that this labeling is more compatible with the gradual degradation assumption of components; therefore, makes it easier for models to be trained. Moreover, if binary prediction of failure is of interest, then after predicting TTE, it can be converted to binary predictions by simple thresholding.

Given that both approaches have pros and cons, the selection of the right one depends on the specifics of the component and the related challenges in the prediction task. There are five well-known causes of turbocharger failure: oil starvation, oil contamination, hot stopping, restricted breathing, and foreign object entry. All of these are mostly gradual processes of turbocharger degradation. Even "foreign object entry" does not usually lead to an abrupt breakdown, but just weakens the internal structure. Therefore, according to the discussion above, we chose the TTE labeling method.

2.3. Censoring

Yet another problem that arises in both of these labeling methods is the censoring. Let's consider the TTE labeling scheme. Measurements logged after the last failure cannot be assigned a label, because we do not know in advance when the next failure is going to happen. There is some research on how to make use of these censored part of the data [20,21]. However, in this work we have decided to neglect these censored part for the sake of simplicity, since we have enough labelled data. Both TTE labeling and censored data concept are shown in Figure 1.

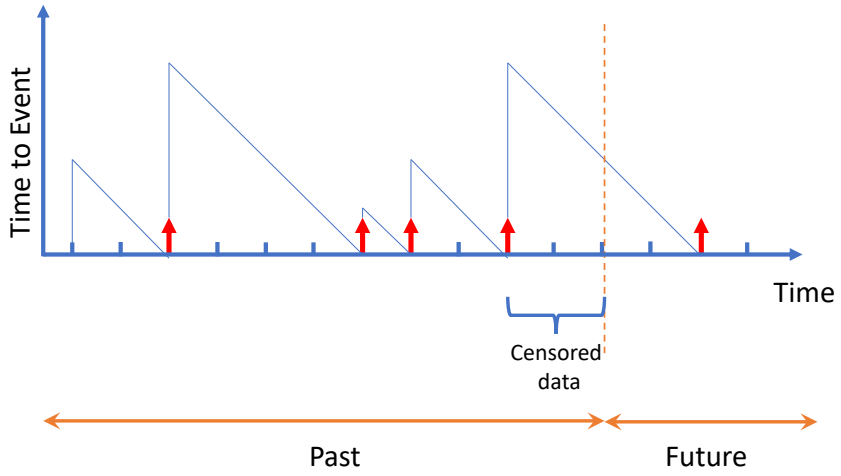

Figure 1. Time to Event (TTE) and censored data.

2.4. Tensor Representation of the Data

Since we will be using RNN-based models, representing data in the form of tensors is of particular interest. A tensor is defined as an N-dimensional array. It allows for an efficient way of representing multidimensional, complex data. For our particular purpose of vehicle predictive maintenance it will suffice to design 3-dimensional tensors which also can be visualized and understood more clearly [22]. The aim is to represent data in the sequential format, to be able to capture the history of each vehicle. Therefore, it is natural to think of one dimension as vehicles, the second dimension as time, and the third dimension as different features.

One challenge is that vehicle history, in principle, is of variable length. We address this by considering a fixed time dependency horizon. For example, if the time dimension is selected to be 10, each instance will contain 10 historical measurements. The output is always the TTE value of the most current measurement; therefore, the architecture of the RNNs is of many-to-one type. The only problem that can arise is for vehicles where the total length of history is smaller than the selected value: in that case zero padding will be applied for missing measurements. It is possible to apply either pre-padding (beginning of the series) or post-padding (end of the series). Each of them is suitable for different scenarios; in particular, in "many-to-one" architectures, the problem with post-padding is that the zeros at the end of the series could possibly cast a shadow over the effect of actual measurements (which comes before zeros). Since there is no such problem for pre-padding it is used in this study.

Figure 2 depicts structures of the designed tensor. There are two advantages in representing the tensors in this format. One is that it is compatible with the way RNNs in Tensorflow [23] consume the input: $(n_{samples}, n_{timesteps}, n_{features})$. The other advantage is that since in this study the number of time steps and number of features are going to be varied (will be discussed in Section 5), this structure makes changing the size of each plane convenient.

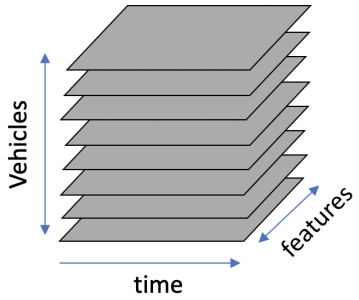

Figure 2. Tensor representation of data.

3. Covariate Shift

A very important challenge in many real world problems, especially for datasets with time series nature, is the covariate shift: the distribution of features (input variables) changes between train and test dataset—or, more generally, across different partitions of the dataset [24]. Formally, covariate shift arises when

$$P_{train}(X) \neq P_{test}(X) \;\; while \;\; P_{train}(Y|X) = P_{test}(Y|X).$$

In other words, change in the joint distribution of X, Y originates from difference in marginal distribution of X_{train}, X_{test}. It is worth mentioning that covariate shift is a subclass of dataset shift. More generally, in addition to marginal distribution shift of features, the conditional distribution of target given features, $P(Y|X)$, can change for different splits of dataset.

As an example of why covariate shift can cause poor generalization, we can consider a scenario where train and test feature values are generated according to two different distributions depicted in Figure 3. Figure 4 shows the generated train and test samples with y-axis as the target values (please note that the distributions in Figure 3 only govern the feature values and not the target values). The black curve would be the target function if we had all data in advance. However, it can be seen that the blue curve learned from training data will result in poor generalization when it comes to test data.

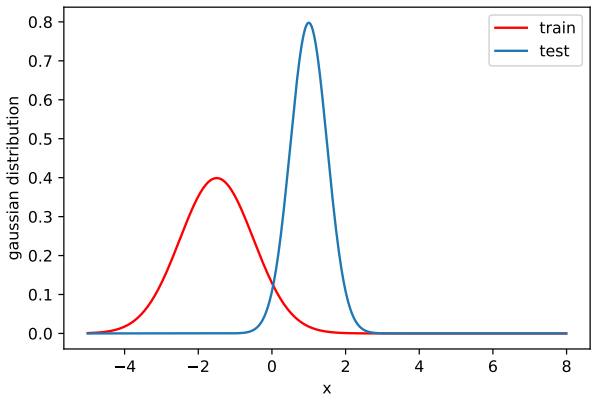

Figure 3. Distributions governing train and test.

In this work, we detect the features with a significant shift, and remove them from training data. It is also possible to apply further transformations to compensate for a shift in those features. Several methods of handling covariate shift exist. For example, Kullback Leibler shift detection method measures the Kullback–Leibler (KL) divergence between splits of a dataset as a measure of distribution change [25]. Statistical test such as Student t-test and Fisher f-test can be used to calculate the shift in mean and variance [26]. There are more recent methods such as Intersection of Confidence Interval rule [27], in which at each point in time a confidence interval is calculated by estimating the standard deviation of a polynomial fitted at the neighborhood of that point. Then, if the confidence interval of a point in time does not have an intersection with the confidence intervals of the previous points in time, a shift will be detected. In [28], control charts which are graphical representations of sample statistics are used for detecting shifts in the mean of time series data. Exponentially weighted average of the previous observations is used to predict one time ahead. Using the predicted value and the control chart, the upper bound and lower bound of upcoming values are calculated. Then, if the observation value of one time ahead falls outside the interval (lower bound to upper bound) a shift will be detected.

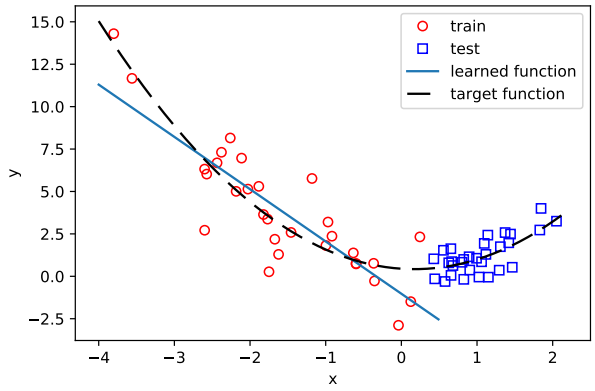

Figure 4. Distributions governing train and test.

For this study, we adopt a method which, in statistical terms, is equivalent to two-sample statistical test. It uses classification accuracy as the test statistic. Intuitively, the idea behind this method is to train a classifier to distinguish between train and test data. The method can, in general, be used to detect covariate shift for combinations of features, i.e., the classifier can be trained and tested on more than one feature. However, for high dimensional data such as ours it is computationally challenging, thus we frame the problem as detection of covariate shift one feature at a time. Algorithm 1 lists the steps of the method. In Algorithm 1, we will use LightGBM [29] model which is a gradient boosting decision tree model as the classifier.

Algorithm 1 Covariate shift detection algorithm

Input:
X ▷ Entire data
tr ▷ Threshold for shift detection
Output:
$cs\_features$ ▷ Set of features detected as high shift

Split X into X_{train} and X_{test}
Add a label column to X_{train} and X_{test} and assign 1 to X_{train} and 0 to X_{test}
Concatenate X_{train} and X_{test} and shuffle to create X_{concat}
Split X_{concat} to create a new X_{train} and X_{test}
for col in $X.columns$ **do**
    $X_{train}^{col} = X_{train}[col]$
    $X_{test}^{col} = X_{test}[col]$
    Train a classifier on X_{train}^{col}
    $test\_auc$ = Calculate AUC on X_{test}^{col}
    if $test\_auc > tr$ **then**
        Add col to $cs\_features$
    end if
end for
return $cs\_features$

4. Model Stacking

Ensemble learning is a machine learning paradigm for improving results by combining several models. The main promises of ensemble learning are reducing variance and bias. From one perspective, ensemble methods can be categorized into two groups: sequential (cascading) and parallel. The basic motivation of sequential methods is to improve upon the residuals of the previous models in the sequence. This category encompasses many machine learning models such as ADAboost, XGboost [30], LightGBM, etc. On the other hand, in a parallel category, models are trained in parallel and the basic idea is to explore independence between models. The more independent they are, the more diverse they will be and hence different representation of data will be learned. This category includes models such as Random Forest, and more generally bootstrap aggregation of machine learning models.

Stacking in its original form is a two-layer architecture. Based on Wolpert terminology [31], first layer data and models are referred as "base" data and models respectively, and second layer cross-validated data and models are referred as "meta" data and models respectively. Stacking is categorized as parallel architecture since base models can be trained independently from each other. The basic architecture of stacking can be shown pictorially using the following diagram (Figure 5).

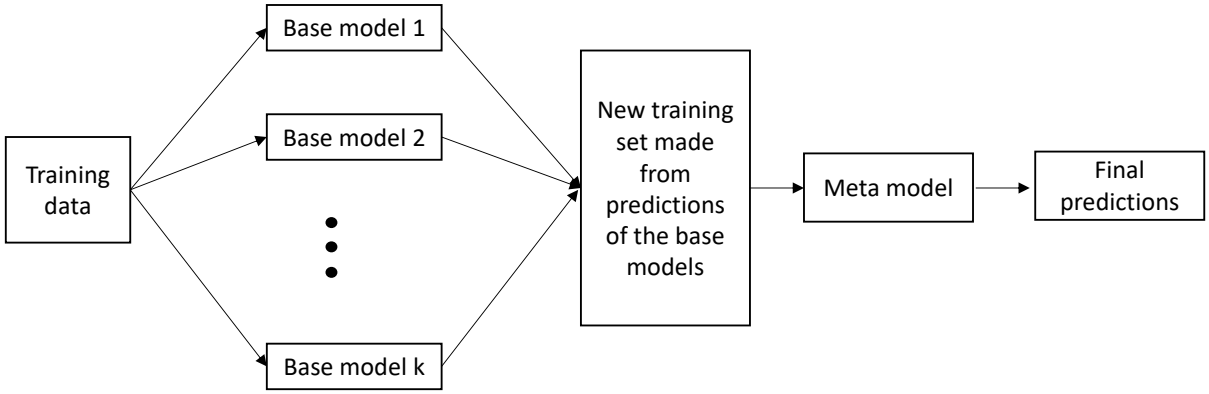

Figure 5. Two-layer stacking architecture.

The benefit of stacking over alternative parallel ensemble approaches lies in the use of complex meta-model that is able to exploit complex patterns in base models diversity. As an intuition, if a base model consistently mis-predicts samples from a region of feature space due to incorrect learning of that region, a meta model may be able to recognize this problem and compensate for it using base models that are better suited for that specific region. Alternative approaches using voting or simple statistical aggregation of predictions are not "aware" of weaknesses or strengths of individual models.

To get the most out of the stacking, base models should be as diverse as possible. More specifically, the predictions of base models should be as decorrelated as possible. Also, it is important to note that only diversity of the base models is not enough, each base model must also produce relatively acceptable predictions. As an extreme case, a combination of a relatively accurate model and a random model, with high probability, will not produce better results, if not worse, than the accurate model.

One important consideration in stacking is to train the models in a way not to leak information about actual labels from base models to meta model. The risk of information leakage originates from the fact that the meta model consumes predictions of the base models as features. In other words, since the base models are provided with the actual labels to be trained on, when they make predictions, the information about actual label will leak to the input of meta model.

To prevent information leakage, the correct way to implement stacking is through k-fold cross validation scheme. The process is shown in Algorithm 2. In words, the main idea is that each base model is trained on k-1 folds and makes a prediction on the holdout fold (untrained fold). This way, the models make predictions on parts of the data which label information was not provided for the models at the training stage.

Note that for each base model, K of them are trained. On the test data, all the saved models for each base model (K of them) make predictions that are then averaged together. This process will be repeated for all base models to generate X_{meta} matrix for the meta model. Finally, the meta model will make the final test prediction.

However, since our data has a time series nature to it, we cannot simply use the standard k-fold cross validation, because if we did, we would end up using the future to predict the past. To tackle this problem we adopt a time-based cross validation in which the data folds that a model is trained on, always have lower indexes than the data folds chosen for making predictions. The time-based cross validation is depicted in Figure 6. Notice that data with index in range [0,n/5] cannot be used in training the meta model since it can cause overfitting.

Algorithm 2 Stacked ensemble algorithm

Input:

X_{train}	▷ Training data
n_{folds}	▷ Number of folds
n_{models}	▷ number of base models
base_models	▷ Set of base learning models
meta_model	▷ Second layer model

Output:

base_models	▷ Trained base learning models
meta_model	▷ Trained second layer model

Split X_{train} into K_{folds} folds
X_{meta} = Create empty array of size $n_{samples} \times n_{models}$
for *model* in *base_models* **do**
    for *i* in K_{folds} **do**
        train *model* on all folds as input excluding *ith* fold
        make prediction on *ith* fold and add to the corresponding rows and column in X_{meta}
    end for
end for
train *meta_model* on X_{meta}
return *base_models*, *meta_model*

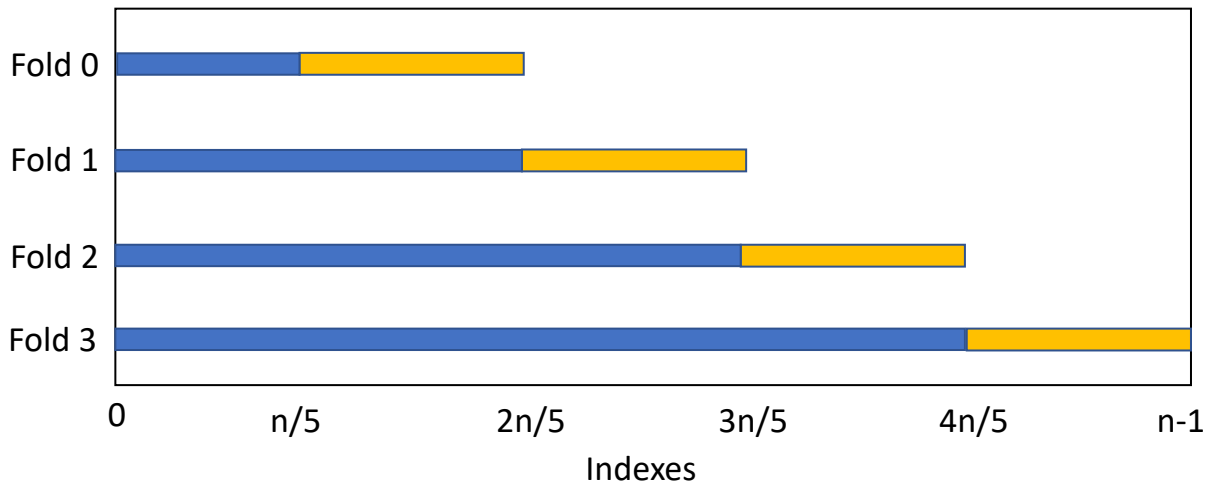

Figure 6. Time-based cross validation.

5. Different Aspects to Maintain Diversity

One of the main goals of this research is to consider different aspects—feature subsets, past dependency resolution, and modeling architecture—to maintain diversity. The idea is that by looking at different aspects, we can get the advantage of each, learn different representations and finally aggregate them to achieve better prediction performance.

The first aspect is "feature subset" by which we mean features will be divided into different subsets and each subset will be the input to the models. By training models on different feature subsets, we encourage each model to focus on different specific aspects of feature space.

Another aspect is the past dependency horizon. Here, past dependency horizon refers to the number of timesteps from the history of measurements to affect the possible future failures. For instance, if the horizon selected to be 10 weeks, it means that for predicting the upcoming events, measurements from 10 last past weeks are considered as the input to a learning model. Note that in this study the most recent measurements are always used for predicting TTE values. An alternative setting would be to predict distant future from past measurements, for instance using $[t-10, t-5]$ to predict TTE value at t.

Horizon will be systematically changed from a smaller number of timesteps to a larger number of timesteps. The idea is by looking at different past horizons, models will capture different aspects of past measurements of the vehicles. One extreme case is to have a horizon of one, which is equivalent to Markovian assumption, hence forcing the model to predict the future only based on the current measurements. The other extreme case is to set the horizon to be the history of the entire measurements, which makes the model take into account all past measurements of the vehicle. However, in this study we are not going to consider these extreme cases. Looking at a very long history makes the problem unrealistic since during the test stage, for new vehicles, we cannot wait for a long time to gather information before making predictions.

The final aspect is the modeling architectures. As mentioned in Section 4, the whole idea of stacking is to train different models with the hope that they can capture different aspects of data. One model might work well on one part of feature space, or one type of failure, while another model can perform well on the others. Ideally, combining such models using a meta model will produce better results.

With these three aspects defined, the process of training base models can be summarized as follows. For each features subset, we iterate over all past dependency horizons, and for each past dependency horizon, we iterate over all modeling architectures and train them. The pseudocode of this process is shown in Algorithm 3.

Features are divided into four different subsets according to the different measurements' meaning. For the proprietary reasons, the division and meaning of the features cannot be disclosed. Also, all features combined is considered as yet another group (before division). Please note that we are talking about the features left after removing high covariate shift features. For past dependency resolution, we consider 5, 10, 15, and 20 timesteps. For modeling part, since the idea is to involve history of the vehicles and treat history as time series, mostly recurrent neural networks are used. One type which is used is LSTM and the other type is Convolutional-LSTM. Therefore, in total because we have 5 feature subsets, 4 past dependency resolutions, and 4 modeling architectures, we will end up with 80 base models. The meta model for the stacking is a simple two-layer Multilayer Perceptron (MLP).

Algorithm 3 Aspect Stacked Ensemble

Input:

X ▷ Data
$feature\_subsets$ ▷ Feature subsets
$time\_horizons$ ▷ list of different past dependency horizons
$Base\_models$ ▷ Set of base learning models

Output:

$Base\_models$ ▷ Trained base learning models

for subset in $feature_{subsets}$ **do**
  $X_{subset} = X[subset]$
  for rs in $time\_horizons$ **do**
    X^{rs}_{subset} = at each time t, concat the previous rs measurements from X_{subset} to from the input at time t
    for model in $base\_model\_architectures$ **do**
      train model on X^{rs}_{subset}
    end for
  end for
end for
return $base\_models$

6. Base and Meta Model Architectures

In this section, the different architectures for base models and the architecture of the meta model are described. For the base models, four different architectures are designed: two LSTM and two CONV_LSTM. There are numerous other architectures that could be chosen; however, to meet the two purposes of performance and diversity these architectures selected in an ad hoc manner.

Table 1 shows the architectures and parameters of the two LSTM models, while Table 2 shows the architectures and parameters of the two CONV-LSTM models. Finally, Table 3 depicts the meta model architecture.

Table 1. LSTM architectures and parameters.

	First Architecture	**Second Architecture**
No. of layers	3	2
Layers type	(LSTM,Dense,Dense)	(LSTM,Dense)
No. of units	(50,20,1)	(100,1)
Units	(relu,relu,linear)	(relu,linear)
Dropout	(0.5,0.2)	(0.5)
Optimizer	ADAM	ADAM
learning rate	0.001	0.001
Cost function	MAE	MAE

Table 2. CONV-LSTM architectures and parameters.

	First Architecture	**Second Architecture**
No. of layers	4	3
Layers type	(1-D CONV,LSTM,Dense,Dense)	(1-D CONV,LSTM,Dense)
No. of CONV filters	128	64
No. of units	(50,20,1)	(100,1)
Units	(relu,relu,linear)	(relu,linear)
Dropout	(0.5,0.2)	(0.5)
Optimizer	ADAM	ADAM
learning rate	0.001	0.001
Cost function	MAE	MAE

Table 3. Meta model architecture and parameters.

No. of layers	2
Layers type	(Dense,Dense)
No. of units	(40)
Dropout	(0.2)
Optimizer	ADAM
learning rate	0.001
Cost function	MAE

7. Experimental Results

In this section, first, the results of applying covariate shift detection method are presented. Then, since we decided to label the data according to TTE labeling, the results of a corresponding regression problem will be reported. The results of stacked ensemble will be compared to simple ensemble aggregation such as average and median of all base models predictions. Also, ensemble result will be compared to the best individual base model result as well. In order to explain the results, we will also analyse model correlations and their relations to prediction errors.

In Algorithm 1, LightGBM model with the parameters reported in Table 4 is used as the classifier for feature selection. Now, we need to find a proper threshold (tr) to compare against LightGBM's AUC for removing features. Different values of (tr) would lead to selection of different subsets of features. For each subset of features obtained for a specific (tr), an LSTM with the Mean Absolute Error(MAE) cost function and the architecture shown in Table 1 is trained. Finally, the obtained MAEs for different values of (tr) are compared to get the best (tr) and features subset.

Table 4. LightGBM setting and parameters.

Objective	binary
Metric	AUC
Boosting	GBDT
Learning rate	0.001
Max depth	7
feature_fraction	0.7
bagging_fraction	0.7

For training the LSTMs, $n_{timesteps}$ in the tensor structure $(n_{samples}, n_{timesteps}, n_{features})$ is 20, and $n_{features}$ comes from the selected threshold for Algorithm 1. The result of this experiment is shown in Figure 7, which indicates threshold of 0.8 as the best choice—which leads to 355 features.

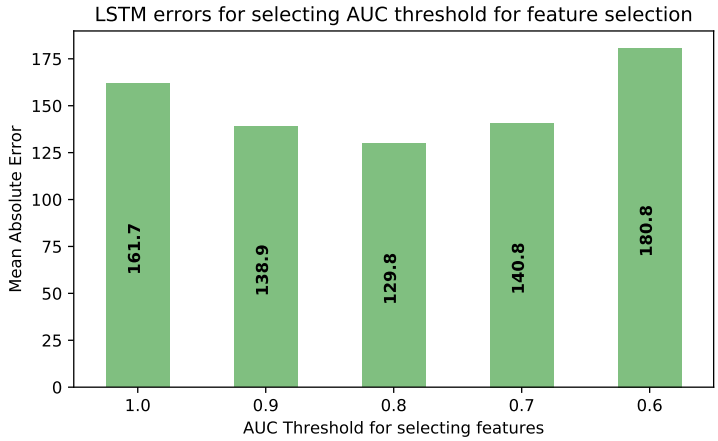

Figure 7. LSTM error results for choosing threshold for feature selection in Algorithm 1.

The problem of predicting time to failure is formulated as a regression task using TTE labeling. Table 5 shows the MAE of different models including ensembles and best individual model. The first row in Table 5 is a very naive estimation of TTE, namely the average TTE across all vehicles. This is included as a baseline for comparison. From Table 5, it can be seen that all ensemble approaches beat the best individual model on test data. The best individual model reported is the first CONV-LSTM architecture defined in Table 2. However, the result of Stacked ensemble outperforms both simpler approaches (mean and median aggregation of the results of base models).

Figures 8 and 9 show the prediction of stacked ensemble predictions and best model prediction against target values, respectively. As can be seen in Figure 9, predictions of the best model generally drop too early in comparison to the target TTE. Also, it can be seen that there are more unnecessary fluctuations in its predictions. On the other hand, the stacked ensemble predictions drop much closer to the target TTE, with not nearly as much fluctuation.

Table 5. Model comparison based on MAE of the TTE predictions.

Model	MAE of TTE Predictions
Mean time to failure (Baseline)	277.17
Best individual model	139.32
Mean of all base models	125.63
Median of all base models	124.26
Stacked ensemble	105.79

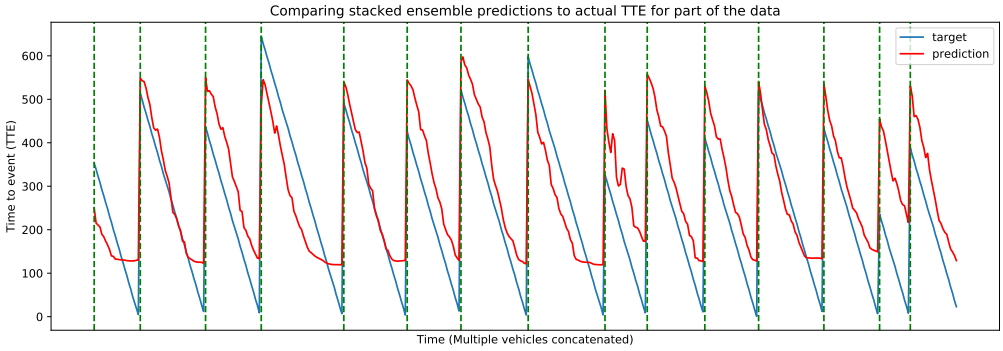

Figure 8. Comparing stacked ensemble results to the target values.

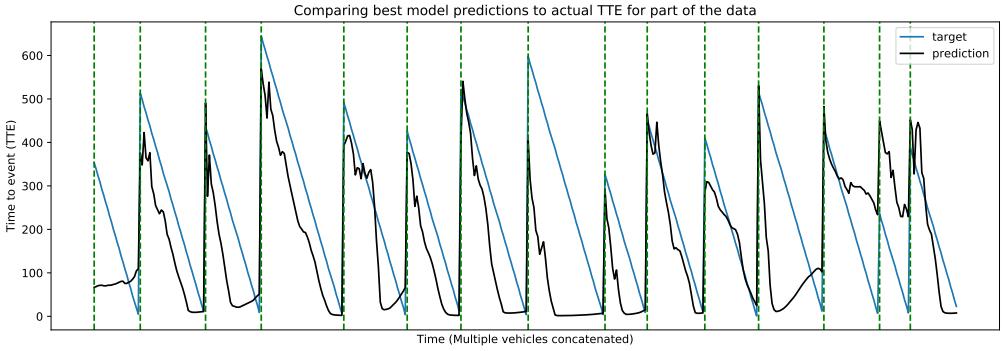

Figure 9. Comparing best model results to the target values.

In order to better visualise the training process of base models in comparison to the meta model, Figures 10 and 11 depict the best base model and meta model learning curve, respectively.

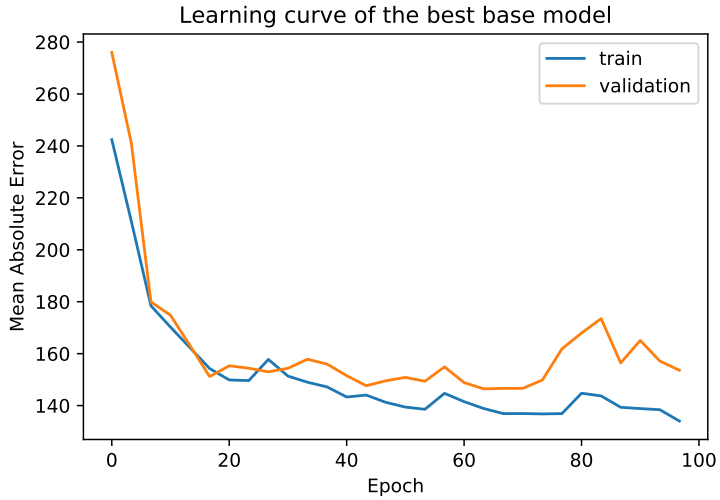

Figure 10. Learning curve of the best base model.

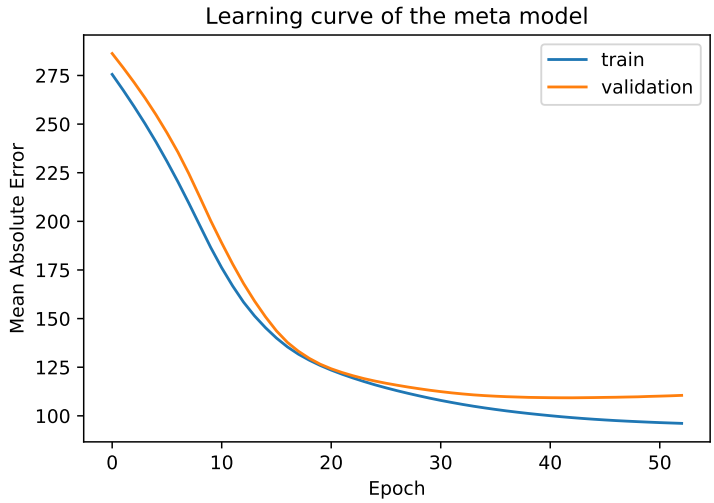

Figure 11. Learning curve of the meta model.

Please note that for both learning curves early stopping is applied (early stopping epoch for best base model is 63 and for meta model is 43). The error on both validation and training sets is lower for meta model, except at the beginning of the curves for the training set. The reason for the error of the base model being lower at the beginning for the training set is that mini-batch training is used for base models, but incremental training is used for meta model. It can also be seen that the evolution of the meta model error is much more stable in comparison to the base model.

As mentioned in Section 4 a sign of diversity in stacking is that the base models' predictions have low correlation with respect to each other. Table 6 depicts some statistics about correlation of base models.

Table 6. Correlation statistics of base models.

Mean correlation of all models	0.56
Max correlation	0.75
Min correlation	0.42

The average correlation of 0.56 is a good indicator of diversity between base models. Please note that in the context of stacking 0.75 correlation (the max correlation) is not still a high correlation given that the models are trying to predict the same target. The maximum correlation is for two LSTM models with the same architecture shown in Table 1 first architecture with same feature subsets but two different past dependency resolution. The lowest correlation is between an LSTM and CONV_LSTM shown in first architecture of Table 1, and second architecture of Table 2 respectively, on two different feature subsets and LSTM with past dependency resolution 5 and CONV_LSTM with past dependency resolution 20. This behaviour can be an indicator of maintaining diversity by looking at different aspects.

In Figure 12 we present how the correlation between base models and the meta model is related to the performance of the former. It can be seen that for base models with between 0.5 and 0.6 correlation to the meta model, the MAE values are almost steady (with only minor fluctuations). After that, as the correlation increases the MAE reduces until the correlation reaches to almost 0.7. From this point on, the MAE values seem to remain steady again. Decrease in MAE as the correlation increases is expected since the meta model is predicting target values; hence, higher correlation between the base models and the meta model means higher predictability and consequently lower MAE.

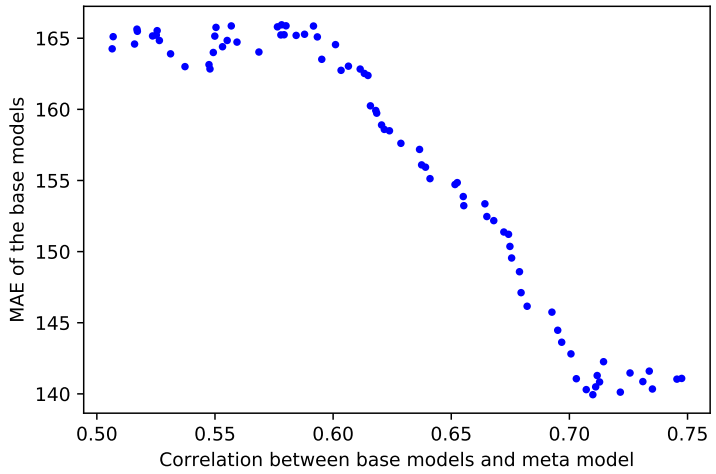

Figure 12. Comparing stacked ensemble results to the target values.

8. Conclusions

In this paper, a stacked ensemble of recurrent neural networks for the task of predictive maintenance is proposed. We address two shortcomings in existing state-of-the-art: time series analysis and considering multiple aspects to build ensembles of diverse models. We demonstrate experimentally, using real-world data from automotive industry, that the proposed approach leads to significant performance improvement.

Framing the data in tensor format with the history of vehicles as a dimension and feeding them to LSTM and CONV-LSTM models allows us to perform time series analysis. Using stacked ensembles is shown to outperform simpler aggregation methods. Finally, we demonstrate how diversity can be tackled by bringing multiple aspects into consideration. These aspects are: different features subspace, different past dependency horizon, and different modeling architectures.

Comparing the stacked ensemble results to the best individual model and other standard ensemble methods shows that the proposed framework leads to a promising boost of performance. The diversity of the base models in the stacked ensemble is verified by calculation the correlation between predictions of the base models.

Author Contributions: P.S.M.: conceptualization, methodology, formal analysis, writing original draft; S.N.: conceptualization, formal analysis, surpervision, review and editing; S.P.: conceptualization, formal analysis, surpervision, review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by VINNOVA.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Prytz, R.; Nowaczyk, S.; Rögnvaldsson, T.; Byttner, S. Predicting the need for vehicle compressor repairs using maintenance records and logged vehicle data. *Eng. Appl. Artif. Intell.* **2015**, *41*, 139–150. [CrossRef]
2. Fan, Y.; Nowaczyk, S.L.; Rögnvaldsson, T.; Antonelo, E.A. Predicting air compressor failures with echo state networks. In Proceedings of the Third European Conference of the Prognostics and Health Management Society 2016, Bilbao, Spain, 5–8 July 2016.
3. Zhao, J.; Xu, L.; Liu, L. Equipment fault forecasting based on ARMA model. In Proceedings of the 2007 International Conference on Mechatronics and Automation, Harbin, China, 5–8 August 2007; pp. 3514–3518.
4. Baptista, M.; Sankararaman, S.; de Medeiros, I.P.; Nascimento, C., Jr.; Prendinger, H.; Henriques, E.M. Forecasting fault events for predictive maintenance using data-driven techniques and ARMA modeling. *Comput. Ind. Eng.* **2018**, *115*, 41–53. [CrossRef]
5. Zhang, X.; Xu, R.; Kwan, C.; Liang, S.Y.; Xie, Q.; Haynes, L. An integrated approach to bearing fault diagnostics and prognostics. In Proceedings of the 2005 American Control Conference, Portland, OR, USA, 8–10 June 2005; pp. 2750–2755.
6. Kamlu, S.; Laxmi, V. Condition-based maintenance strategy for vehicles using hidden Markov models. *Adv. Mech. Eng.* **2019**, *11*, 1687814018806380. [CrossRef]
7. Wang, X.; Ma, L.; Wang, B.; Wang, T. A hybrid optimization-based recurrent neural network for real-time data prediction. *Neurocomputing* **2013**, *120*, 547–559. [CrossRef]
8. Chen, Z.; Liu, Y.; Liu, S. Mechanical state prediction based on LSTM neural netwok. In Proceedings of the 2017 36th Chinese Control Conference (CCC), Dalian, China, 26–28 July 2017; pp. 3876–3881.
9. Chen, K.; Pashami, S.; Fan, Y.; Nowaczyk, S. Predicting Air Compressor Failures Using Long Short Term Memory Networks. In Proceedings of the EPIA Conference on Artificial Intelligence, Vila Real, Portugal, 3–6 September 2019; pp. 596–609.
10. Kovalev, D.; Shanin, I.; Stupnikov, S.; Zakharov, V. Data mining methods and techniques for fault detection and predictive maintenance in housing and utility infrastructure. In Proceedings of the 2018 International Conference on Engineering Technologies and Computer Science (EnT), Moscow, Russia, 20–21 March 2018; pp. 47–52.
11. Bruneo, D.; De Vita, F. On the use of LSTM networks for Predictive Maintenance in Smart Industries. In Proceedings of the 2019 IEEE International Conference on Smart Computing (SMARTCOMP), Washington, DC, USA, 12–15 June 2019; pp. 241–248.
12. Baraldi, P.; Mangili, F.; Zio, E. A Kalman Filter-Based Ensemble Approach With Application to Turbine Creep Prognostics. *IEEE Trans. Reliab.* **2012**, *61*, 966–977. [CrossRef]
13. Seera, M.; Lim, C.; Nahavandi, S.; Chu Kiong, L. Condition monitoring of induction motors: A review and an application of an ensemble of hybrid intelligent models. *Expert Syst. Appl.* **2014**, *41*, 4891–4903. [CrossRef]
14. Wu, H.; Meeker, W.Q. Early detection of reliability problems using information from warranty databases. *Technometrics* **2002**, *44*, 120–133. [CrossRef]
15. Dong, G.; Liu, H. *Feature Engineering for Machine Learning and Data Analytics*; CRC Press: Boca Raton, FL, USA, 2018.
16. Lei, X.; Sandborn, P.A. Maintenance scheduling based on remaining useful life predictions for wind farms managed using power purchase agreements. *Renew. Energy* **2018**, *116*, 188–198. [CrossRef]

17. Sivalingam, K.; Sepulveda, M.; Spring, M.; Davies, P. A Review and Methodology Development for Remaining Useful Life Prediction of Offshore Fixed and Floating Wind turbine Power Converter with Digital Twin Technology Perspective. In Proceedings of the 2018 2nd International Conference on Green Energy and Applications (ICGEA), Singapore, 24–26 March 2018; pp. 197–204.
18. Si, X.S.; Zhang, Z.X.; Hu, C.H. Data-Driven Remaining Useful Life Prognosis Techniques. In *Springer Series in Reliability Engineering*; Springer: Berlin/Heidelberg, Germany, 2017.
19. Lei, Y. *Intelligent Fault Diagnosis and Remaining Useful Life Prediction of Rotating Machinery*; Butterworth-Heinemann: Oxford, UK, 2016.
20. Allik, B.; Piovoso, M.J.; Zurakowski, R. Recursive estimation with quantized and censored measurements. In Proceedings of the 2016 American Control Conference (ACC), Boston, MA, USA, 6–8 July 2016; pp. 5130–5135. [CrossRef]
21. Allik, B.; Miller, C.; Piovoso, M.J.; Zurakowski, R. The Tobit Kalman Filter: An Estimator for Censored Measurements. *IEEE Trans. Control Syst. Technol.* **2016**, *24*, 365–371. [CrossRef]
22. Gardner, J.; Koutra, D.; Mroueh, J.; Pang, V.; Farahi, A.; Krassenstein, S.; Webb, J. Driving with Data: Modeling and Forecasting Vehicle Fleet Maintenance in Detroit. *arXiv* **2017**, arXiv:1710.06839.
23. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Available online: tensorflow.org (accessed on 18 December 2019).
24. Sugiyama, M.; Krauledat, M.; MÃžller, K.R. Covariate shift adaptation by importance weighted cross validation. *J. Mach. Learn. Res.* **2007**, *8*, 985–1005.
25. Sugiyama, M.; Suzuki, T.; Nakajima, S.; Kashima, H.; von Bünau, P.; Kawanabe, M. Direct importance estimation for covariate shift adaptation. *Ann. Inst. Stat. Math.* **2008**, *60*, 699–746. [CrossRef]
26. Montgomery, D.C. *Introduction to Statistical Quality Control*; John Wiley & Sons: Hoboken, NJ, USA, 2007.
27. Alippi, C.; Boracchi, G.; Roveri, M. A just-in-time adaptive classification system based on the intersection of confidence intervals rule. *Neural Netw.* **2011**, *24*, 791–800. [CrossRef] [PubMed]
28. Raza, H.; Prasad, G.; Li, Y. EWMA model based shift-detection methods for detecting covariate shifts in non-stationary environments. *Pattern Recognit.* **2015**, *48*, 659–669. [CrossRef]
29. Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; Ma, W.; Ye, Q.; Liu, T.Y. Lightgbm: A highly efficient gradient boosting decision tree. In Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 4–9 December 2017; pp. 3146–3154.
30. Chen, T.; Guestrin, C. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference On Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 785–794.
31. Wolpert, D. Stacked Generalization. *Neural Netw.* **1992**, *5*, 241–259. [CrossRef]