machine learning & knowledge extraction

Article

Black Box Adversarial Reprogramming for Time Series Feature Classification in Ball Bearings' Remaining Useful Life Classification

Alexander Bott *,†, **Felix Schreyer** †, **Alexander Puchta** and **Jürgen Fleischer**

wbk Institute of Production Science, Karlsruhe Institute of Technology (KIT), Kaiserstraße 12, 76131 Karlsruhe, Germany
* Correspondence: alexander.bott@kit.edu; Tel.: +49-1523-950-2643
† These authors contributed equally to this work.

Abstract: Standard ML relies on ample data, but limited availability poses challenges. Transfer learning offers a solution by leveraging pre-existing knowledge. Yet many methods require access to the model's internal aspects, limiting applicability to white box models. To address this, Tsai, Chen and Ho introduced Black Box Adversarial Reprogramming for transfer learning with black box models. While tested primarily in image classification, this paper explores its potential in time series classification, particularly predictive maintenance. We develop an adversarial reprogramming concept tailored to black box time series classifiers. Our study focuses on predicting the Remaining Useful Life of rolling bearings. We construct a comprehensive ML pipeline, encompassing feature engineering and model fine-tuning, and compare results with traditional transfer learning. We investigate the impact of hyperparameters and training parameters on model performance, demonstrating the successful application of Black Box Adversarial Reprogramming to time series data. The method achieved a weighted F1-score of 0.77, although it exhibited significant stochastic fluctuations, with scores ranging from 0.3 to 0.77 due to randomness in gradient estimation.

Keywords: transfer learning; RUL prediction; ball bearings; black box model; time series classification; predictive paintenance

check for updates

Citation: Bott, A.; Schreyer, F.; Puchta, A.; Fleischer, J. Black Box Adversarial Reprogramming for Time Series Feature Classification in Ball Bearings' Remaining Useful Life Classification. *Mach. Learn. Knowl. Extr.* **2024**, *6*, 1969–1996. https://doi.org/10.3390/make6030097

Academic Editor: Laura Po

Received: 27 June 2024
Revised: 19 August 2024
Accepted: 22 August 2024
Published: 27 August 2024

1. Introduction

Machine Learning (ML) has come a long way from the first perceptron to today's transformer networks. While predictions suggest that the ML market will surpass USD 500 billion by the end of the second decade of the 21st century [1], current ML models share one problem: their dependence on sufficiently large, high-quality datasets.

The subfield of transfer learning (TL) offers a remedy for this problem. TL involves using knowledge from established tasks to improve learning for new but related tasks [2]. Typically, these approaches are concerned with fine-tuning models obtained from the source domain, especially in the highly successful field of deep learning.

However, fine-tuning requires complete understanding and access to the pre-trained model. Models that are not accessible, so-called black box models, can, therefore, often not be incorporated into TL, regardless of their performance potential.
In light of this challenge, this paper investigates a novel TL approach called Black Box Adversarial Reprogramming, which permits using black box models.

In 2018, Elsayed et al. presented a new type of adversarial attack in their paper "Adversarial Reprogramming of Neural Networks" [3]. With their so-called adversarial reprogramming algorithm, the authors showed that the function of neural networks can be misappropriated. Adversarial reprogramming can, therefore, be used to hijack third-party resources. However, it turns out that the novel concept also has positive potential.

Two years after the publication of Elsayed et al., the authors Tsai et al. presented their work "TL without Knowing: Reprogramming Black-box ML Models with Scarce Data and Limited Resources" [4]. In this paper, the authors propose applying the adversarial reprogramming algorithm to solve problems that usually fall within the scope of TL. They show that adversarial reprogramming can solve tasks in domains where only a few labeled data are available by reusing a model trained on another domain. The fact that their method can be used with black box algorithms makes their work particularly innovative. They, therefore, refer to their method as Black Box Adversarial Reprogramming (BAR). Tsai et al. demonstrate the effectiveness of the BAR algorithm in image classification but believe that their approach can be successfully applied to other types of problems as well.

This paper builds on the knowledge previously obtained by [3,4]. The paper aims to investigate the potential of the BAR algorithm when applied to time series data. This investigation is conducted in the context of predictive maintenance. In this context, data scarcity is especially limiting. The lifetime data of machine components need either long-term observations in productive machines or relatively costly investigations on specified test benches. This is why singular and task-specific ML models are not a realistic option for various components.

The data used in the paper are sensor data from rolling bearings used in production plants [5]. The data are run-to-failure data, on the basis of which the RULs of the respective bearings are to be determined.

Notably, this work is limited to traditional ML; deep learning approaches are not considered. Traditional ML approaches have several advantages, such as a higher level of explainability and better uncertainty estimation capabilities.

In order to evaluate the novel BAR approach, a conventional TL model is developed to serve as a benchmark and allow for a performance classification of the new approach. The work includes an evaluation of the two approaches on a common test dataset. The analysis is qualitative in nature.

2. Transfer Learning

The application areas of ML are diverse, ranging from topics such as pattern recognition to those like machine vision [1]. However, ML architectures share a common problem. In order to fully unleash their potential, they need sufficiently large datasets [6]. In certain domains of application, it may not be feasible, or hardly so, to generate datasets of adequate quality. Consequently, conventional ML methods are deemed non-feasible. However, the concept of TL can provide a solution. TL, a sub-field of Machine Learning (ML), entails leveraging knowledge from established tasks to enhance the learning process for new but related tasks [2].

2.1. Definitions

In order to incorporate the definition of TL as outlined in the current literature, it is necessary to first understand two underlying definitions. The concept of *domain* is one of the definitions that must be comprehended, along with the definition of *task*.

- **Domain**

A domain $\mathcal{D}$ consists of two components, a feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$, where $X = x_1, x_2, \ldots, x_n \in \mathcal{X}$ [7].

- **Task**

Given a specific domain, $\mathcal{D} = \{\mathcal{X}, P(X)\}$, a task consists of two components, a label space $\mathcal{Y}$ and an objective function $f(\cdot)$, which cannot be observed but can be learned from training data. These training data consist of pairs $\{x_i, y_i\}$ with $x_i \in X$ and $y_i \in Y$. A task is denoted as $\mathcal{T} = \{Y, f(\cdot)\}$ [7].

Understanding the two terms, task and domain, allows one to address the definition of TL commonly used in the literature.

- **Transfer Learning**

Given a source domain $\mathcal{D}_\mathcal{S}$, a learning task $\mathcal{T}_\mathcal{S}$, a target domain $\mathcal{D}_\mathcal{T}$m and a learning task $\mathcal{T}_\mathcal{T}$, transfer learning (TL) aims to help improve the learning of the target predictive function $f_T(\cdot)$ in $\mathcal{D}_\mathcal{T}$ using knowledge in $\mathcal{D}_\mathcal{S}$ and $\mathcal{T}_\mathcal{S}$, where $\mathcal{D}_\mathcal{S} \neq \mathcal{D}_\mathcal{T}$ or $\mathcal{T}_\mathcal{S} \neq \mathcal{T}_\mathcal{T}$ [7].

Thus, TL describes a process in which knowledge from a previously learned task $\mathcal{T}_\mathcal{S}$ in a domain $\mathcal{D}_\mathcal{S}$ is leveraged to then improve learning and increase learning efficiency in another domain $\mathcal{D}_\mathcal{T}$ and/or task $\mathcal{T}_\mathcal{T}$. Further, the definition of TL indicates that the term is not particularly restricted. Unsurprisingly, TL can, therefore, be employed in a variety of application scenarios. The field of application of TL can thus quickly become confusing, which is why a compact taxonomy of the field follows below. There are a number of ways to categorize the different TL approaches. In this paper, we follow the taxonomic structure used by Zhuang et al., which groups approaches according to problem categorization and solution categorization [8].

2.2. Problem Categorization

The definition of TL states that either the domains themselves or the tasks on the domains must differ to be considered TL ($D_S \neq D_T$, or $T_S \neq T_T$). Subcategories of TL can be identified based on the (in)consistency between domains and/or tasks. Such a categorization approach is also referred to as *Label-Setting-Based Categorization* [7].

It should be noted, however, that domains and tasks may differ, and a certain degree of relationship must be maintained to use TL effectively.

2.3. Solution Categorization

Focusing on solutions, also called implementation approaches, provides an alternative method to problem-based categorization. In this context, the emphasis lies in distinguishing the various approaches based on their technical concepts. Once again, it was the authors Pan and Yang who identified four different key categories, which are briefly presented below [7].

2.3.1. Instance-Based

The instance-based approach involves assigning individual weights to instances derived from the source domain. By applying these weights, it becomes possible to perform importance sampling or, alternatively, to directly incorporate the weighted instances into the learning function of the model [9]. Numerous studies deal with the instance-based approach [10–16].

2.3.2. Feature-Based

One goal of the feature-based approach and the instance-based approach is to reduce the disparity between the marginal and conditional distributions of the source and target domains. Where the feature-based concept differs is that it revolves around discovering a meaningful feature representation tailored to the target domain. The pre-existing knowledge is consequently embedded within this new feature representation [7–9]. There are also a large number of papers that examine the feature-based approach [4,17–24].

2.3.3. Parameter-Based

The parameter-based approach deviates from the previously introduced methods regarding its fundamental concept. It operates on the assumption that there exist shared model parameters or parameter distributions between the source and target tasks. In this context, knowledge transfer is encapsulated within these parameters or parameter distributions [7–9]. Papers examining this approach include the following [25–35].

2.3.4. Relational-Based

The relational-based approach is particularly well suited for facilitating TL across relational domains. The underlying premise is that specific relationships within the data remain consistent across diverse domains. Consequently, relational-based approaches aim to efficiently transfer the relationships inherent in the data [7–9]. This approach has been less studied in the literature to date [36–38].

Certainly, not all approaches can be clearly categorized into a single group. Algorithms frequently blend concepts from various categories, resulting in them being referred to as hybrid approaches [9].

3. Predictive Maintenance

Maintenance strategies have been extensively studied in research, presumably not least due to their high economic impact [9,39]. These strategies are typically classified into three subcategories, *Reactive Maintenance*, *Preventive Maintenance*, and *Predictive Maintenance* (PdM), as documented in the literature [9,40]. PdM stands out through the use of advanced analytics and ML. Combined with continuous monitoring, PdM leverages these technologies to assess an asset's health and predict when maintenance will be required. The primary benefits associated with using PdM comprise a reduction in unplanned downtime, increased system reliability, and reduced operating costs by optimizing the allocation of maintenance resources. A comparison of the cost patterns of predictive maintenance with different maintenance strategies can be found in [41]. For a graphical illustration of the strategic differences between predictive, reactive and preventive maintenance, see Figure 1 [9,40].

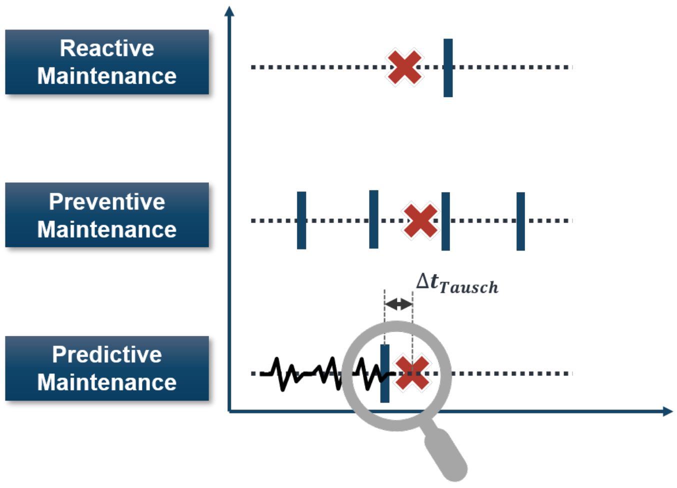


Figure 1. Conceptual differences between different maintenance strategies. (After: [41])

Various approaches to implementing PdM concepts exist. Data-driven methods stand out, particularly when combined with ML. In recent years, these models have gained importance due to the increased availability of computing power and the abundance of data. The models extract information from data to contextualize the degradation state of components, the health of the system, or its Remaining Useful Life (RUL).

Transfer Learning in Predictive Maintenance

As this paper focuses on the intersection of transfer learning and predictive maintenance, this section will specifically look at relevant work from this subdiscipline. The topic has gained considerable attention, as evidenced by the existence of several meta-studies. Ran et al. have dedicated a separate section to transfer learning in predictive maintenance [40]. Azari and colleagues conducted a systematic literature review on transfer learning for predictive maintenance in Industry 4.0 [9]. Another review, by Zheng and colleagues, focuses on cross-domain fault diagnosis [42].

A substantial number of scientific studies depend on deep learning, including the study by [43]. The authors tackle the problem of data typically not being independent and identically distributed. They suggest the application of a Contractive Denoising Autoencoder to synthesize more stable features. Subsequently, these features function as input to a support vector machine (SVM) for RUL prediction. Wu and colleagues address the problem of varying feature distributions in source and target domains as well but specifically focus on machine operational data and the impact of prior working condition distributions on them. The authors propose the utilization of convolutional neural networks (CNNs) and Long Short-Term Memory Networks (LSTMs) to address the issue. Wen and colleagues examine feature distribution between domains, focusing on feature extraction [44]. They note that deep learning can extract features automatically, simplifying the typically challenging feature engineering process. Additionally, the team presents a method using a sparse autoencoder to extract features automatically, with the peculiarity that the auto-encoder can be utilized for prediction when enhanced with a softmax layer as well. Zhu and colleagues further address the issue of feature distribution [45]. Specifically, they tackle the issue by employing a hidden Markov model to automatically identify the fault occurrence time. To tackle the actual distribution problem, they utilize a multilayer perceptron to identify domain invariant features. Da Costa et al. address feature distribution in the context of parameter-based transfer learning and suggest a technique that involves freezing parameters to minimize the number of parameters that require adjustment [46]. The proposed approach utilizes LSTM and a Domain Adversarial Neural Network. Xia et al. focus on transferring information from fault datasets to predict RUL via transfer learning [47]. Looking at the input features, Ma et al. find that relying solely on the first principal component of a PCA on the characteristic features leads to better performance than using high-dimensional features [48]. The research presented by [49] focuses on the demands of online fault detection and diagnosis. Numerous CNNs are employed by the authors to tackle the issue. Similarly, Cheng et al. utilize CNNs to address instance selection from the source domain [50]. Additionally, Ong et al. concentrate on the ideal allocation of resources, including human capital, for predictive maintenance in an industrial IoT environment. DRL is utilized for this purpose [51]. A Transfer from Virtual to Real Machine (TVRM) scenario is discussed in [52]. A Serial Stacked Autoencoder and deep neural network are deployed in their work. In addition, research is focused on achieving optimal training of DL techniques in terms of accuracy and training speed [53,54].

What all of these publications share is a reliance on deep learning methods. While such methods offer benefits such as automated feature engineering, they also carry inherent shortcomings. To work well, deep learning methods require extensive amounts of data [6]. However, in the context of PdM, data are a rare commodity, particularly run-to-failure data [9,42].

Moreover, deep learning models are inherently opaque. Due to their complex non-linear structure, these models are commonly deployed with limited transparency, revealing little information on the process of how they arrive at their predictions [55]. In particular, these models provide no meaningful insight into the uncertainties underlying their predictions.

However, in a production setting, decisions made by models have a high economic relevance, making both the traceability of the decision process and the associated uncertainty a valuable asset that is gaining attention in academic research [45,56].

Approaches based on traditional Machine Learning can offer a solution in this regard. Because of their less complex structure, they offer a higher level of transparency. In addition, they can be built with much fewer data [6,57].

Mahyari and Locher discuss a traditional ML-based PdM approach for industrial robots. They observe that a separate model is usually required for each task the robot performs. Thus, they recommend identifying a joint feature space for these tasks by employing the manifold alignment algorithm [58]. However, their proposed approach only involves the generation of the joint feature space, without incorporating actual PdM methods.

Another traditional Machine Learning approach is the one of Mao and colleagues [59]. They tackle the problem of varying feature distributions with classical ML models. Therefore, in the initial step, they employ hierarchical clustering to identify similar degradation patterns and obtain the pivot feature set of the respective instances. The obtained set is subsequently used as input to an SVM.

Shen and Yan recommend utilizing so-called intermediate domain data when dealing with low-quality features in order to improve RUL prediction [60]. Intermediate domain data refer to time series data that does not reach the point of failure. The authors propose utilizing an SVM for the RUL prediction as well.

Although the methods of Mao et al. and Shen and Yan largely avoid the aforementioned weaknesses of deep learning-based approaches, they have their own drawbacks. Part of this is the authors' decision to rely on SVMs. SVMs belong to the traditional TL approaches. As such, they require fewer data and are more transparent than their DL counterparts. However, SVMs are not among the most transparent approaches, especially for high-dimensional input features. At the same time, SVMs tend to encounter problems when the size of the input data exceeds a certain threshold. This threshold is about 100,000 data points [61]. However, in production environments with a lot of relevant sensor data, this limit may be exceeded. Lastly, when it comes to the increasingly important issue of uncertainty estimation [45,56], SVMs are not the ideal approach either. While they can provide such an estimate [62], their estimate is not as statistically sound as, say, ensemble-based random forests [63].

Furthermore, all of the approaches presented in Table 1 fail to address black box models. All presented approaches require the presence of white box models that can be accessed or retrained to enable transfer learning on them. Thus, circumstances where only black box models are available are excluded from transfer learning. As a result, for such scenarios where black box models are present, only complex, costly, and possibly infeasible new implementations of ML models are possible [64,65].

Table 1. Literature at the intersection of TL and predictive maintenance. ■ Deep learning, □ Traditional ML.

Use Case	Focus	ML Method	Deep Learning	Year	Source
Meta Studies	Generalistic		■	2019	[40]
	Cross-Domain Fault Diagnosis		■	2019	[42]
	Industry 4.0		■	2023	[9]
Prediction	Fault Datasets	CNN, LSTM	■	2021	[47]
	Feature Distribution	LSTM, DANN	■	2020	[46]
	Feature Distribution	CDAE, SVM	■	2019	[43]
	Feature Distribution	LSTM, CNN	■	2019	[66]
	Feature Distribution and FOT	HMM, MLP	■	2020	[45]
	Multi Feature Fusion	ConvNeXt	■	2022	[48]
	Sample Selection	CNN	■	2023	[50]
	Feature Distribution	HCA, SVM	□	2021	[59]
	Low-Quality Features	SVM	□	2021	[60]
Diagnosis	Feature Distribution and Extraction	SAE, DNN	■	2019	[44]
	Simulation to Real World	SSAE, DNN	■	2019	[52]
	Training	NN	■	2019	[53]
	Training	CNN	■	2019	[54]
Detection	Resource Allocation	NN	■	2022	[51]
	Robot Tasks	-	□	2021	[58]
Detection and Diagnosis	Real Time Requirements	CNN	■	2020	[49]

4. Experimental Design

This paper includes the development of two TL approaches based on time series data to study the novel Black Box Adversarial Reprogramming approach. One approach is based on the BAR concept, and the second is based on a conventional TL method. Therefore, the conventional approach serves as a basis for comparison and is referred to as the baseline model. The methodology is shown in Figure 2.

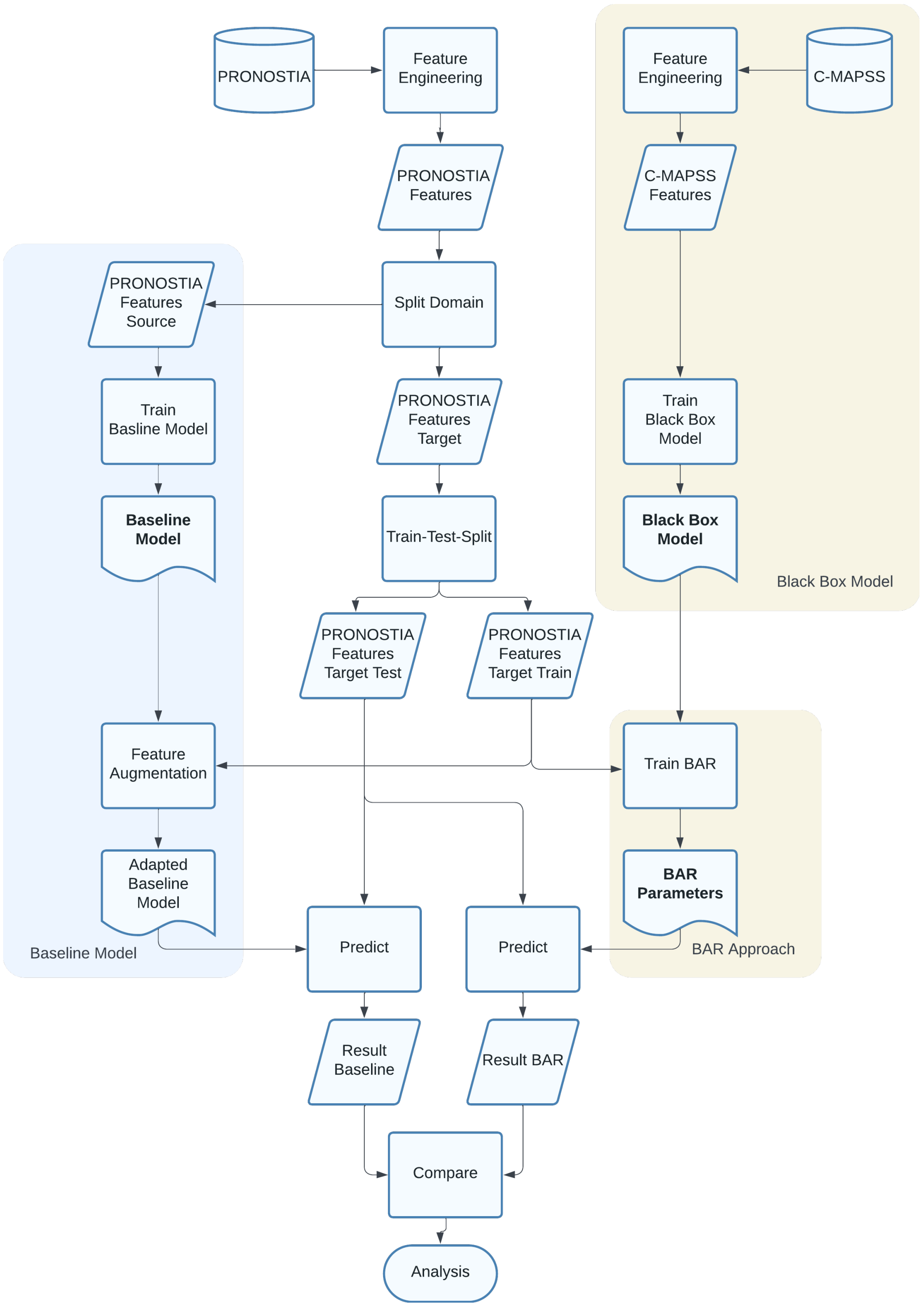


Figure 2. Methodology of this paper.

4.1. Datasets

The time series data used in this thesis are sensor data obtained from existing datasets. This section examines what datasets are being used, why they were selected, and how they are being further processed.

The dataset at the center of this work is the PRONOSTIA dataset [5]. It contains accelerometer data of rolling bearings. These data are run-to-failure data. Accordingly, the dataset is well suited for RUL prediction and classification. The dataset was chosen because it is rather extensive compared to alternatives; i.e., it includes multiple different test runs. In addition, three different experimental settings are examined in the dataset, which is essential in the context of transfer learning.

Based on these three different settings, the dataset is divided into a source domain $\mathcal{D}_S$ and a target domain $\mathcal{D}_T$. The first two experimental settings are considered the source domain $\mathcal{D}_S$. The third setting is considered the target domain $\mathcal{D}_T$.

The baseline model is later fitted to the source domain $\mathcal{D}_S$ of the dataset. The target domain $\mathcal{D}_T$ of the dataset is the domain to which the knowledge is to be transferred for both transfer learning approaches. Data of both the source and data target domains are divided into training and test sets (see Figure 3). To achieve a balanced ratio of training and test instances, the split is based on the author's judgment.

The NASA Turbofan Degradation dataset, also known as the C-MAPSS dataset, is the second dataset used in this thesis [65]. It contains sensor data from turbofan engines. The dataset was selected because it addresses the issue of predicting the Remaining Useful Lifetime of a device as well. At the same time, the domain gap between the PRONOSTIA and the C-MAPSS dataset is sufficiently large.

C-MAPSS is divided into four parts, each containing a different combination of aircraft engine settings. Only the first subset is used in this thesis. The subset contains data from only one engine setting. Wanting to limit complexity is driving this choice. The C-MAPSS dataset is used to train the black box model. The adversarial reprogramming algorithm is then based on this black box model.

Unlike with the PRONOSTIA dataset, the train–test split of the C-MAPSS dataset happens at random. Twenty-five percent of the dataset is reserved for testing, and the other seventy-five percent is part of the training dataset.

The structure of the two datasets used in this thesis is shown graphically in Figure 3.

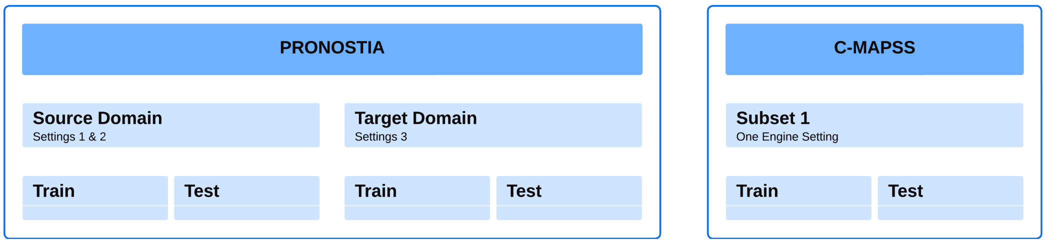


Figure 3. Structuring of the datasets used in this paper.

4.2. Models and Transfer Learning Approaches

This work's traditional ML algorithms require manual feature engineering. The library *tsfresh* is used because it specializes in time series feature engineering and implements various features from different domains. Whether features are relevant and subsequently selected is decided based on the training data (source domain) through a supervised feature selection via hypothesis testing. The resulting features X are input to the ML models. Random forests are chosen for the different source domain predictions as ML models because of their efficiency and suitability for uncertainty estimation in future investigations. The different models are used for the baseline approach and transfer scenarios.

The baseline TL model adopts the *adapt* framework for feature-based TL by [18]. This method was chosen to exploit the easy integration of expert knowledge through features that are beneficial in environments with limited data. This approach was implemented to provide a performance baseline against which the results of the novel BAR approach can be related. The BAR algorithm, distinct yet based on a similar random forest model, innovates by learning translation functions. These functions adapt input features from the target domain and output probabilities from the source model, enabling meaningful data processing across different contexts.

In the transfer scenario of the BAR, a significant domain gap necessitates using the *C-MAPSS* dataset to train the Black Box Model. This model is then applied to predict the RUL-Classes of the third condition in the *PRONOSTIA* dataset, similar to the baseline approach.

4.3. Analysis

The analysis of the BAR approach is based on evaluating the two developed TL approaches and the different transfer scenarios. The same target dataset is used for this evaluation, namely the subset of the PRONOSTIA dataset $X_{\mathcal{D}_T}^{PRONOSTIA}$, i.e., its target domain data. The approaches are all trained and evaluated on these data.

Based on the evaluation, a qualitative analysis of the performance of the BAR algorithm is conducted. Specifically, the performance of both TL approaches in the target domain $\mathcal{D}_T$ is investigated. As the algorithm contains a stochastic element, the extent to which this element affects the final performance of the BAR approach is also examined. To this end, the distribution of performance under identical conditions is studied.

In addition to pure performance, the influence of the algorithm's hyperparameters is investigated, too. Both their impact on performance and performance variance are assessed.

Finally, the parameters obtained by training the algorithm are examined. The parameters are hierarchically clustered to investigate whether the BAR training concept can lead to similar parameter patterns, i.e., to the same minima of the optimization problem.

5. Baseline Approach

the following subsection describes the baseline approach against which the BAR approach is compared.

5.1. Preprocessing

The data are scaled using *scikit-learn*'s *RobustScaler*, which employs a median-based method using the interquartile range to minimize the influence of outliers. This is crucial as the PRONOSTIA dataset contains outlier data that may be critical for Remaining Useful Life (RUL) prediction.

Following scaling, a rolling window approach is applied to generate subwindows from the time series, artificially increasing the number of instances for training the ML model. Each window w starts with the beginning of data collection at $t_0 = 0$ and covers a minimum period of 20 min. Each window w_i is 15 min longer than the previous window w_{i-1}. Thus, the first window w_1 covers exactly 20 min, the second window w_2 covers 35 min, the third window w_3 covers 50 min, and so on. The last window w_n, therefore, covers the entire time series.

To determine the RUL, two terms must first be explained.

- **Total Useful Lifetime**

The *Total Useful Lifetime* describes the *entire* service life of a bearing up to the point where a failure occurs and a maintenance case arises.

- **Pro Rata Useful Lifetime**

The *Pro Rata Useful Lifetime* describes the *remaining* service life of a bearing up to the point where a failure occurs and a maintenance case arises. In particular, this value differs from the total useful lifetime in that it is determined from any point in the time series. For a time window, the pro rata useful lifetime is determined by the end of the time window. Both total useful lifetime and pro rata useful lifetime are provided in units of time.

Given those two terms, the RUL and subsequently the RUL classes can then be determined. The RUL is calculated by taking the quotient of pro rata useful lifetime and total useful lifetime.

$$\text{RUL} = \frac{\text{Pro Rata Useful Lifetime}}{\text{Total Useful Lifetime}} \tag{1}$$

$$\overline{\text{RUL}} = 1 - \text{RUL} \tag{2}$$

Classification can then be performed based on the obtained RULs. The classification approach used in this thesis is based on the work of Xia and colleagues [67]. They present an RUL classification approach that revolves around (3).

$$((1 - 2^{-(i-1)}) \cdot 100\%, (1 - 2^{-(i)}) \cdot 100\%] \tag{3}$$

All in all, this formula can be used to classify the windows into n different classes, $n \in \mathbb{N}^+$. i denotes the respective class, $0 < i \leq n$.

For this thesis, the value $n = 3$ was chosen, motivated by the fact that a limited amount of data are available and to keep the complexity low. This results in three classes, which can be found in Table 2. Note that (3), which is used for classification, refers to the intervals of $\overline{\text{RUL}}$ and not to those of RUL.

Table 2. RUL classes of the PRONOSTIA dataset according to (3).

	1	**2**	**3**
$\overline{\text{RUL}}$ interval of class i	(0, 50]	(50, 75]	(75, 100]
RUL interval of class i	[100, 50)	[50, 25)	[25, 0)

The PRONOSTIA dataset is segmented into sub-datasets by operational condition, split into training and testing sets, where scenarios 1 and 2 form the source domain $\mathcal{D}_S$ and scenario 3 is the target domain $\mathcal{D}_T$.

5.2. Feature Engineering

This study explores the feature extraction process for Remaining Useful Life (RUL) prediction using the Python-based ML library, *tsfresh*.They identify computationally efficient and relevant features. Due to the size, an auxiliary dataset, $D^{train}\mathcal{D}_S$, aux, comprising every tenth data point, was utilized to manage the computational load. Feature relevance was determined through a statistical analysis where the 20 features with the lowest p-values were selected for each RUL class, indicating high statistical significance. This selection process aimed to optimize the balance between computational efficiency and the inclusion of statistically significant features.

5.3. ML Model

The ML model of the baseline approach is based on a random forest classifier from *scikit-learn* [68]. Random forests belong to the traditional ML models. They are composed of an ensemble of decision trees. The prediction of a random forest is generated from the individual predictions of the decision trees. The ML concept is interpretable, robust, efficient, and well suited for uncertainty estimation [69–71]. The model is trained on the source features of the PRONOSTIA dataset.

As with other ML models, the choice of hyperparameters has a fundamental impact on the predictive performance of a random forest model. In this paper, a combination of Bayesian search and grid search was used. Thus, in this paper, the parameter space was first restricted using a Bayesian search. Six Bayesian searches of 150 iterations each were performed on a large parameter space determined by expert judgment. Based on those results, the parameter space was reduced and a grid search was performed on the reduced space.

The metric used in all searches is the *macro f1 score*. The use of the *f1 score* is motivated by the fact that the harmonic mean of *precision* and *recall* provides a more balanced representation of the model performance. In particular, the *macro f1 score* was implemented because it is well suited for unbalanced data. A natural consequence of the RUL classification scheme used in this paper is that such unbalanced data will occur.

The calculation steps for the determination of the *macro f1 score* are as follows.

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} \tag{4}$$

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \tag{5}$$

$$f1_i = 2 \cdot \frac{\mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} = \frac{\mathrm{TP}}{\mathrm{TP} + \frac{1}{2}(\mathrm{FP}+\mathrm{FN})} \tag{6}$$

$$f1_{macro} = \frac{1}{N} \cdot \sum_{i=1}^{N} f1_i \tag{7}$$

This implementation incorporates class weights to address the issue of imbalanced data further. When appropriate hyperparameters are identified, minority classes are weighted more. Thus, choosing hyperparameters will ensure that minority classes are well classified.

5.4. Transfer Learning

The TL approach used in the baseline model belongs to the area of feature-based TL approaches. More precisely, the concept applied is a feature augmentation approach based on [18]. In order to implement this concept, the open-source library *ADAPT* [72] was used.

The feature augmentation method presented by Daumé III decomposes input features into three parts. The transformation is handled differently depending on whether the input features are part of the source or target domain.

$$\Phi^S(x) = \langle x, x, 0 \rangle \tag{8}$$

$$\Phi^T(x) = \langle x, 0, x \rangle \tag{9}$$

$\Phi^{S/T}(x)$ describes mapping functions with 0 as the zero vector. This method transforms the domain adaptation problem into a standard supervised learning problem by augmenting the feature space of both the source and the target data.

Imagine you have a model trained to detect road signs from photos taken in clear, daytime conditions (source domain). Now, you want this model to recognize the same road signs from photos taken in various challenging conditions, such as at night or during adverse weather (target domain), where thereare fewer labeled data available.

In this approach, each feature in the original data (e.g., shape, color, size) is expanded into three versions: a general version, a source-specific version, and a target-specific version. For instance, the color feature might have three versions: a general color feature applicable to all conditions, a daytime-specific color feature, and a night-specific or adverse weather-specific color feature.

The augmented data are then used to train the model. The training data from the clear daytime conditions (source domain) include both the general and daytime-specific color features, while the limited data from the night or adverse weather conditions (target domain) include both the general and night-specific or adverse weather-specific color features. This helps the model learn which color characteristics are universally relevant (e.g., stop signs are red) and which characteristics are specific to the lighting or weather conditions (e.g., how red appears at night versus during the day).

Feature augmentation is a supervised TL approach suitable for regression and classification tasks.

Initially, the target data $X_{\mathcal{D}_T}^{train}$ are provided to the feature augmentation method of *ADAPT* along with the already trained baseline model. The method then computes the augmented features according to the methodology presented above. The final step is to refit the model to the augmented features.

6. Black Box Adversarial Reprogramming

This section discusses the Black Box Adversarial Reprogramming approach. First, the black box model on which the reprogramming approach is based is presented. Then, the novel adversarial reprogramming TL approach takes center stage.

6.1. Preprocessing

The feature engineering process for the C-MAPSS dataset is similar to that of the PRONOSTIA dataset. However, of the four different subsets, only subset 1 is incorporated in this paper. The subset contains only one operational setting. By using only this subset, the complexity of the black box model is kept to a minimum. The calculation of the RULs of the window instances is identical to the calculation of the RULs of the instances of the PRONOSTIA dataset. However, there is one difference. Instead of three RUL classes, four different classes defined by expert judgment are considered for RUL classification in the case of the C-MAPSS dataset. Table 3 shows the exact breakdown of the resulting RUL classes.

Table 3. RUL classes for the C-MAPSS dataset.

	1	**2**	**3**	**4**
$\overline{\text{RUL}}$ interval of class i	(0, 50]	(50, 70]	(70, 90]	(90, 100]
RUL interval of class i	[100, 50)	[50, 30]	[30, 10]	[10, 0]

6.2. ML Model

Like the baseline approach, the black box ML model undergoes feature extraction, yielding 110 features. Employing a random forest framework selected for its interpretability and computational efficiency, implementation is facilitated through *scikit-learn*. Bayesian optimization is utilized for hyperparameter tuning due to its conceptual superiority and computational efficiency over grid search. The parameter space for optimization is expertly defined, and after 250 iterations, the optimal hyperparameter configuration is determined. Model evaluation relies on accuracy, considering a slightly expanded set of feature classes compared to the baseline, ensuring balance. A weighted class representation is integrated into the training process to address residual class imbalances.

6.3. Adversarial Reprogramming Algorithm

6.3.1. Functional Structure

BAR describes an algorithm that can hijack the functionality of an existing ML model and repurpose it for another, unrelated task. It does not matter whether the hijacked model is a black box or a white box model. The task for which the model is being hijacked is different from the task of the underlying black box model, in the domain in which it runs ($\mathcal{D}_S \neq \mathcal{D}_T$) and/or in the task itself ($T_S \neq T_T$). Therefore, the application scenario described falls within the research area of TL.

The BAR algorithm's mechanism can be divided into three parts. A graphical representation of the operating principle is also shown in Figure 4.

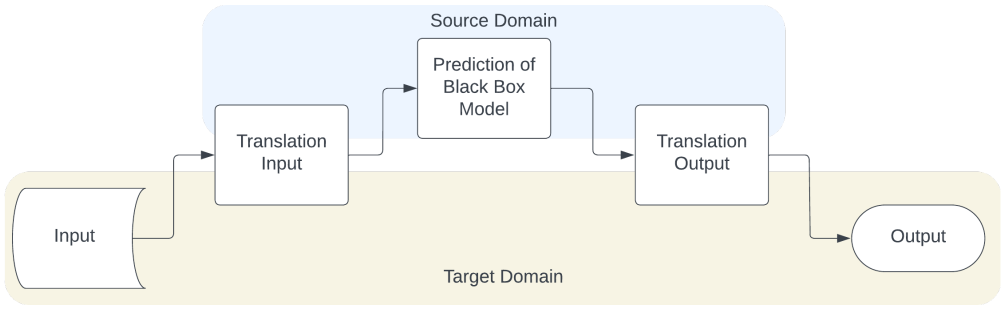


Figure 4. Principle of operation of the BAR algorithm.

- **Translation Input**

Initially, the algorithm receives the input features of the target domain X_T. These features need to be processed so that they can be used as input to the black box model. This processing is referred to as *translation*. As it is the translation of target features, the process is also called *input translation*. The translation is achieved in three steps.

First of all, the input features X_T are scaled. This is conducted using *scikit-learn's* MinMaxScaler. By compressing all features to the interval [0,1], the dimensions in which each feature exists can be aligned, and a finer-grained response to the subsequent translation step is possible.

In this paper's implementation, the target domain's input features are the target domain features of the PRONOSTIA dataset.

In this subsequent step, the input features X_T are multiplied by a weight vector W; see (10). Thus, the input features of the target domain X_T are adapted so that they can be processed in the source domain $\mathcal{D}_S$ in a meaningful way. The procedure for obtaining the weight vector will be explained in Section 6.3.2.

$$\hat{X}_T = X_T * W \tag{10}$$

For the third step, note that the features may differ between X_T and X_S. Thus, for the black box model to handle the target feature vector X_T, the dimensions of the target feature vector X_T must be aligned with X_S. In this implementation, $dim(X_T) << dim(X_S)$ holds. Accordingly, columns must be added to the target feature vector to allow the black box model to process X_T. For this purpose, columns containing only the value 0 are added. There is also another technicality to consider. The column names of X_T must be aligned for the black box model to process the feature vector.

In this work, X_T contains 43 features while X_S contains 110, so 67 columns had to be added.

- **Prediction of the Black Box Model**

The translated features can then be passed to the black box model. The model processes the input ordinarily and returns a prediction corresponding to the RUL class probabilities. The output of the model is denoted by $\hat{y}$.

- **Translation Output**

Output $\hat{y}$ then has to be translated back into the target domain $\mathcal{D}_T$. To accomplish this, ref. [4] use a method called *multi-label mapping* in their paper. Multi-label mapping describes a procedure in which the output probabilities of multiple source classes are aggregated to form the probability of a single target class. This function is also referred to as *k-to-1 mapping function* $h(\hat{y})$. The formula for calculating the probability $h_j(\cdot)$ of a single target class j can be taken from (11). S describes a subset of k source labels on which the mapping is performed.

$$h_j(\hat{y}) = \frac{1}{|S|} \sum_{s \in S} \hat{y}_s \tag{11}$$

The prerequisite for applying this method is that the number of classes in the source task T_S is significantly greater than the number of classes in the target task T_T.

In this implementation, however, there are four source classes ($\hat{y} \in \mathbb{R}^{4\times1}$), and three target classes ($y \in \mathbb{R}^{3\times1}$). As a result, the version of the multi-label mapping method described above is not a good fit.

Thus, the version of multi-label mapping implemented in this paper is as follows.

$$y_1 = \hat{y_1}, y_2 = \hat{y_2}, y_3 = \hat{y_3} + \hat{y_4} \tag{12}$$

As (12) shows, translation does not affect the first two classes. However, the third class is formed by aggregating the third and fourth classes of the source domain $\mathcal{D}_S$.

The BAR algorithm shown in Listing 1 can be successfully executed after obtaining the output vector y for the original input X_T through the following functional structure.

Listing 1. Three functions comprise the functional structure of the BAR algorithm.

```
def translate_X(X, W):
    # Apply MinMaxScaler to Columns
    X = X.scale_columns()
    # Align number of columns and rename them
    X = X.align_and_rename_columns()
    # Translate X to X_hat using W
    X_hat = X * W
    return X_hat

def predict_y(X_hat, model)
    # Obtain model predictions on X_hat
    y_hat = model.predict_probabilities(X_hat)
    return y_hat

def translate_y(y_hat):
    # Aggregate the last two columns
    y_hat[:, 2] += y_hat[:, 3]
    # Delete the redundant last column
    y_hat = np.delete(y_hat, 3, axis=1)
    # return translated output
    return y
```

6.3.2. Training Process

As with most ML methods, the BAR approach must undergo training before it can be used meaningfully. The fact that the algorithm works with black box models poses a challenge. Unlike most BAR approaches, the algorithm does not have access to the underlying BAR model during the training process and, in particular, does not have access to internal model gradients.

To this end, zeroth-order optimization is applied, analogous to the work of [4]. This optimization technique is unique because it estimates the model's gradient without access to the underlying model.

The learning procedure of the BAR algorithm presented here is based on the principle of minimizing a loss function using a gradient estimate based on zeroth-order optimization. In a figurative sense, the algorithm learns how to translate features of the target domain $\mathcal{D}_T$ to the source domain $\mathcal{D}_S$ in a way that leads to meaningful results. Training can be divided into several sub-steps, some repeated in loops. All training steps are presented below. For a graphical overview of the steps in the learning process, see Figure 5.

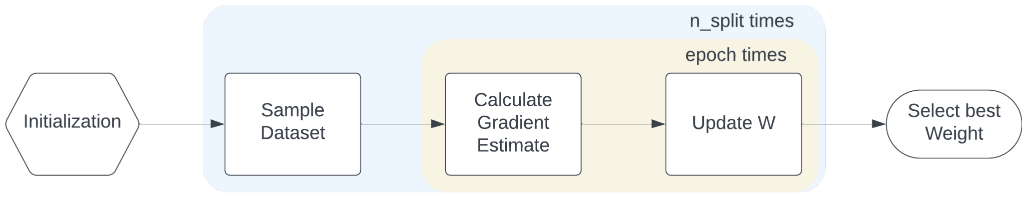


Figure 5. Structure of the training process of the BAR algorithm.

- **Initialization**

Training begins with an initialization step. This step involves the target features X_T and the weight vector W. Since the target feature vector X_T may contain a different number of features than the source feature vector X_S, the first step is to align X_T to X_S dimensionally. Second, the feature columns are scaled. Scaling is performed with the help of *scikit-learn's* MinMaxScaler. Third, the column names of the target vector X_T must be matched to those of the source vector X_S. This is necessary for the black box model to process the features without error.

Further, initializing the weight vector W is part of initialization as well. The vector is initialized as one vector in the format $W = [1, 1, \dots, 1] \in \mathbb{R}^{1\times n}$.

- **Sample Dataset**

The sampling of the dataset follows the initialization step. The dataset is sampled to compensate for its imbalance. Since a good performance is desired for both the majority and minority classes, a rebalancing of the classes is sought. Oversampling and undersampling are applied in alternating order. The number of loops this process goes through is specified by the parameter *n_split*. In the implementation, sampling is performed with the help of the library *imbalanced-learn* [73].

- **One-Sided Averaged Gradient Estimator**

Sampling is then followed by the step of gradient estimation. The concept of the *one-sided averaged gradient estimator* is used for this purpose. The estimator averages several estimated gradients at a given point $W^{(i)}$. The gradient estimates are a function of the loss function $\mathcal{L}$.

- **Loss Function**

As in most BAR optimization methods, the loss function is at the core of zeroth order optimization. In this paper, the loss function consists of two components.

The first component is the regular cross-entropy loss $\mathcal{L}_{CE}$. In the context of BAR, the loss function $\mathcal{L}_{CE}$ penalizes any deviations between the probability prediction p_i and the truth label y_i.

$$\mathcal{L}_{CE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij}) \tag{13}$$

The second part of the loss function is referred to as the penalty term $\mathcal{L}_{PEN}$. The term computes the sum of the differences between the probability the model assigns to the true class $p_{ij}|y_{ij} = 1$ and the highest probability that the model assigns to any class $\max_{j\in J} p_{ij}$ for each instance i. Thus, the term partially counteracts the cross-entropy loss by only penalizing incorrect predictions. There is no penalty for correct predictions, even if the corresponding prediction probability may be rather low.

$$\mathcal{L}_{PEN} = \sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} \cdot (\max_{j\in J} p_{ij} - p_{ij}) \tag{14}$$

Ergo, the loss function $\mathcal{L}$ consists of the cross-entropy loss $\mathcal{L}_{CE}$ and the penalty loss $\mathcal{L}_{PEN}$. The penalty loss is provided with a weighting factor δ. δ balances the two loss functions.

$$\mathcal{L} = \mathcal{L}_{CE} + \delta \cdot \mathcal{L}_{PEN} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij}) + \delta \cdot \sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} \cdot (\max_{j \in J} p_{ij} - p_{ij}) \tag{15}$$

The gradient is then calculated using the loss function $\mathcal{L}$ as follows.
The computation starts at a point $W^{(i)}$, wherein i denotes the current training iteration, $i \in \mathbb{N}$. At that point, $W^{(i)}$, q random vectors U_j are generated. A simple gradient g_j is then calculated along each of these q vectors U_j according to (16).

$$g_j = (\mathcal{L}(W + U_j) - \mathcal{L}(W)) \cdot U_j \tag{16}$$

Then the arithmetic mean of the q different gradients U_j is calculated and the estimated gradient $\overline{g}$ is determined.

$$\overline{g} = \frac{1}{q}\sum_{i=1}^{q} g_j \tag{17}$$

- **Update W**

Using the estimated gradient $\overline{g}$ obtained at point $W^{(i)}$, the weights of the weight vector $W^{(i)}$ can then be updated. Two update functions were examined for this purpose: a linear (18) and an exponential one (19).

$$W^{(i+1)} = W^{(i)} + \alpha \cdot \overline{g} \tag{18}$$

$$W^{(i+1)} = W^{(i)} + e^{-decay \cdot iteration} \cdot \alpha \cdot \overline{g} \tag{19}$$

The linear update function works rather straightforwardly. The weight vector $W^{(i)}$ is adjusted in the direction of the gradient $\overline{g}$. This adjustment takes place at learning rate α. The larger the learning rate, the larger the adjustment step.

The exponential update function undergoes an exponential adaptation of the learning rate during training. Its evolution is determined by the *decay rate* and the learning rate α.

Ultimately, the linear update function (18) provided a more stable behavior for the BAR algorithm. For this reason, the linear update function is used in this implementation. The exponential alternative remains a subject of future research.

The steps of gradient estimation and subsequent weight updating are performed together iteratively. The goal is to find an optimal weight vector W on the sampled dataset. The *epochs* parameter is the hyperparameter that describes the number of repetitions.

- **Select best W**

Training ends after $n = n\_splits \cdot epochs$ iterations. The weight vector $W^{(i)}$, $0 < i < n$, with the smallest loss $\mathcal{L}$ is selected as the final weight vector W.

6.3.3. Hyperparameters

At this point, the hyperparameters adjusted as part of the training process are listed and introduced separately. There are six different hyperparameters. For an overview of the hyperparameters, see Table 4.

Table 4. Overview of hyperparameters that can be adjusted during the training process.

Learning Rate	**Number Resamplings**	**Weighting Penalty Term**	**Number Random Vectors**	**Iterations per Sampling**	**Size of Vectors** q
α	*n_splits*	δ	q	*epochs*	*vector_size*

The hyperparameter α represents the so-called *learning rate*. The parameter describes the step size by which the weight vector $W^{(i)}$ is adjusted in the direction of the gradient $\overline{g}$.

n_splits specifies the number of times the underlying feature set X_T is sampled during training.

The two terms of the loss function $\mathcal{L}$ are balanced by the parameter δ. The higher the value of this parameter, the greater the influence of the penalty term $\mathcal{L}_{PEN}$. This, in turn, means that the cross entropy loss $\mathcal{L}_{CE}$ becomes less important. As a result, more emphasis is placed on making the correct class prediction and less on ensuring this is completed with high confidence.

q specifies the number of random vectors U_j to be formed for the estimation of the gradient estimate $\overline{g}$. The *vector_size* parameter describes how large the q randomly generated vectors U_j are. The larger the parameter, the larger the vectors generated.

epochs, in turn, describes how many iterations the weight adjustments go through for each sampled dataset.

Three hyperparameters directly affect the computational complexity of the training process. Those parameters are *n_splits*, q, and *epochs*.

7. Results

7.1. Baseline

This section examines the performance of the baseline approach. First, the performance of the underlying ML model is analyzed in its source domain $\mathcal{D}_S$. Then, the performance of the baseline approach in the target domain $\mathcal{D}_T$ is examined. Both approaches are evaluated predominantly on the basis of the *macro f1 score*. The *macro f1 score* averages the *f1 scores* of all individual classes, thus emphasizing the minority classes.

7.1.1. Performance Source Domain

The random forest model that the baseline approach is based on was fitted to the source domain features X_S of the PRONOSTIA dataset.

The model performs particularly well on the training data. Out of the 159 training instances, the model correctly classifies 152 instances. Notably, the model performs very well in the minority classes.

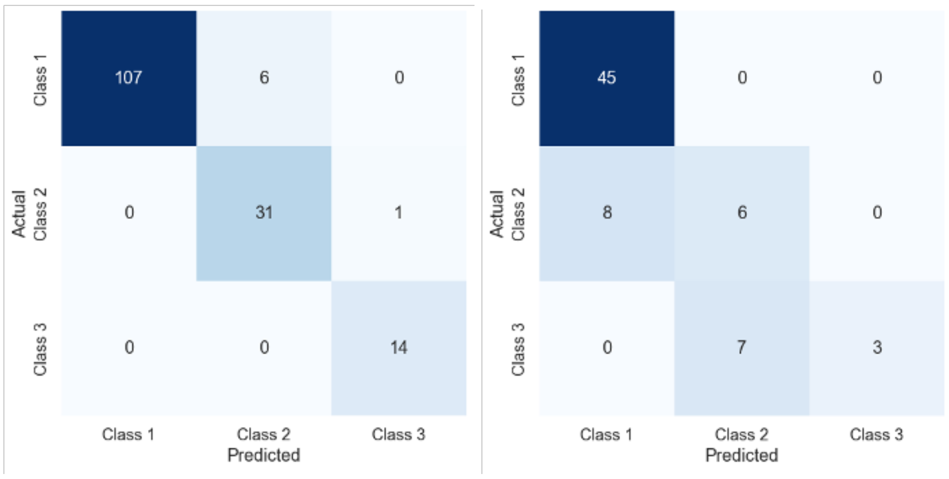
Based on the test data, the performance of the model is decent but significantly worse. Out of 69 instances, the model correctly classifies 54 instances. However, the model has major problems with minority classes 2 and 3, with less than 50% of them being correctly classified in each case. The resulting *macro f1 score* of 0.61 also captures this observation. In comparison, the model achieves a *macro f1 score* of 0.95 on the training data (see Table 5 and Figure 6).

Table 5. Various baseline model performance metrics in the source domain

	Train Data				**Test Data**			
	Precision	**Recall**	**f1 score**	**Support**	**Precision**	**Recall**	**f1 score**	**Support**
Class 1	1.00	0.95	0.97	113	0.85	1.00	0.92	45
Class 2	0.84	0.97	0.90	32	0.46	0.43	0.44	14
Class 3	0.93	1.00	0.97	14	1.00	0.30	0.46	10
accuracy			0.96	159			0.78	69
macro avg	0.92	0.97	0.95	159	0.77	0.58	0.61	69
weighted avg	0.96	0.96	0.96	159	0.79	0.78	0.76	69

Thus, based on the resulting numbers alone, it is reasonable to assume that the baseline model is overfitting. However, its hyperparameter selection should not allow excessive overfitting, as shown by the parameters used in Section 5.3. The hyperparameters were chosen rather restrictively to avoid overfitting, with a minimum of eight samples for a leaf node and a minimum offive samples to split an internal node. To ensure that the model would work on the target domain, merely the maximum tree depth was chosen to be on the high side with 5.

In summary, the model performs very well on the training data and solidly on the test data as seen in Table 5.



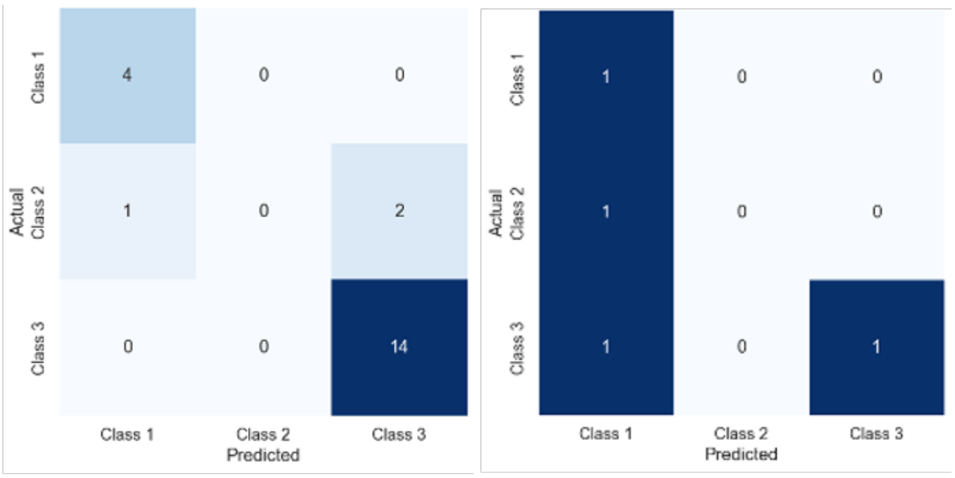
(**a**) Confusion matrix on train data. (**b**) Confusion matrix on test data.

Figure 6. Confusion matrices of the baseline model in the source domain.

7.1.2. Performance Target Domain

It must be adapted to the latter to enable the previously developed random forest to also work on the target domain. The random forest is fitted to the features obtained through feature augmentation to achieve this.

In general, the performance pattern in the target domain is very similar to the pattern in the source domain as seen in Table 6. The model performs quite well on the training data. However, even with the training data, the model does not perform well with the minority classes, in contrast to the source domain. More specifically, the model predicts class 2 fundamentally incorrectly. Classes 1 and 3, however, are classified reliably as seen in Figure 7.



(**a**) Confusion matrix on train data. (**b**) Confusion matrix on test data.

Figure 7. Confusion matrices of the baseline model in the target domain.

Table 6. Various baseline model performance metrics in the target domain.

	Train Data				Test Data			
	Precision	**Recall**	**f1 score**	**Support**	**Precision**	**Recall**	**f1 score**	**Support**
Class 1	0.80	1.00	0.89	4	0.33	1.00	0.50	1
Class 2	0.00	0.00	0.00	3	0.00	0.00	0.00	1
Class 3	0.88	1.00	0.93	14	1.00	0.50	0.67	2
accuracy			0.86	21			0.50	4
macro avg	0.56	0.67	0.61	21	0.44	0.50	0.39	4
weighted avg	0.74	0.86	0.79	21	0.58	0.50	0.46	4

On the test data, however, the performance of the model is poor. Only half of the instances are classified correctly. In particular, the model tends to be biased towards incorrectly predicting Class 1. The poor performance is also reflected in a low *macro f1 score* of 0.39 (see Table 7). The baseline model still achieves a macro f1 score of 0.61 on the training data.

Table 7. Final hyperparameter configuration.

Hyperparameter	**Search Space**	**Hyperparameter**	**Search Space**
α	0.6	q	50
n splits	5	epochs	120
δ	7	vector size	135

Overall, the performance of the baseline approach shows that while the model works well in the source domain, it has difficulty in the target domain. Based on the training data, the model encounters difficulties with one of the minority classes. This observation also holds for the test data. Note, however, that the test data consist of only four instances. For our investigation, the performance of the training data is especially relevant because it enables an assessment of the capability to find a suitable mapping to the target domain. Our future research will investigate the ability to generalize and evaluate through the test set.

7.2. Black Box Adversarial Reprogramming

This section presents and evaluates the performance of the BAR approach. First, the underlying black box model and its performance are discussed. Then, the performance of the BAR algorithm on the target domain is analyzed.

7.2.1. Performance Black Box Model

The black box model is fitted to the features extracted from the C-MAPSS dataset (see Section 4.1). Thus, unlike the baseline model, the black box model must be attributed to four different RUL classes. The black box model shows stable performance across train and test data. However, the model has some difficulty distinguishing between classes that are directly adjacent to each other. Regardless, no gross misclassifications are found in the train or the test data. The stable performance is also reflected in the model's macro f1 scores, which differ slightly at 0.85 on the train data and 0.79 on the test data (see Table 8).

Table 8. Various black box model performance metrics for training and test data in the source domain.

	Train Data				Test Data			
	Precision	**Recall**	**f1 score**	**Support**	**Precision**	**Recall**	**f1 score**	**Support**
Class 1	0.93	0.94	0.93	82	0.81	0.94	0.87	18
Class 2	0.76	0.90	0.83	63	0.86	0.75	0.80	24
Class 3	0.77	0.84	0.80	97	0.53	0.85	0.65	20
Class 4	0.96	0.76	0.85	101	0.98	0.75	0.85	53
accuracy			0.86	343			0.79	115
macro avg	0.86	0.86	0.85	343	0.79	0.82	0.79	115
weighted avg	0.86	0.85	0.85	343	0.85	0.80	0.81	115

7.2.2. Hyperparameters

The BAR algorithm's hyperparameter selection and retrieval process are discussed before its performance is analyzed. Given the stochastic nature of the BAR training process involving randomly generated vectors U_j, this stochastic element necessitates consideration during hyperparameter determination. Consequently, a three-stage grid search is employed to finalize the parameter configuration. Each hyperparameter configuration undergoes repeated training sessions to accommodate the stochastic training process. The grid search unfolds as follows:

1. Initial coarse grid search: Each configuration is computed three times.
2. Subsequent finer grid search: Each configuration is computed seven times.
3. Final fine grid search: Each configuration undergoes eleven training sessions.

The metric employed for hyperparameter selection is the average macro *macro f1 score* across all training sessions. In cases where scores exhibit similarity, consideration is given to the variance of performance, with preference accorded to scores demonstrating lower variance. The hyperparameters *n_splits* and *epochs* are determined independently of the grid search based on expert estimation. Given their combined influence on training iterations and the constraints of this study, these parameters are generously estimated to expedite the grid search process.

7.2.3. Performance Black Box Adversarial Reprogramming

It shall be recalled that the approach's training process is stochastic since the vectors used for gradient estimation are randomly generated. Correspondingly, the resulting weight vector W is also subject to this stochastic element, affecting the approach's performance.

Therefore, the first step is to examine the extent to which this affects the final performance of the approach. The training process was run 360 times for the final hyperparameter configuration, as seen in Table 7. Figure 8 shows a histogram of the obtained *macro f1 scores*. It can be seen that the resulting performance varies greatly. Results below 0.3 as well as above 0.7 can be found.

The results should be evaluated in relation to the baseline model in particular. As a reminder, the baseline model achieved a *macro f1 score* of 0.61 on the target domain train data. It can be seen in Figure 8 that of the 360 training runs that were performed using the BAR approach, only 13 were able to perform as well as or better than the baseline model.

The best macro f1 score achieved with the BAR algorithm on the target train data is 0.77, as shown in Table 9. In this case, the BAR approach outperforms the baseline model. Compared to the baseline model's matrix, we can see that the BAR approach is a bit more consistent. For example, it can sometimes correctly predict class 2 but sometimes mispredicts class 3. On the test data, however, the BAR approach performs slightly worse on the *macro f1* metric with a score of 0.27 as seen in Table 9. Again, the small size of the test dataset should be noted.

Table 9. Various performance metrics of the best BAR approach in the target domain

	Train Data				Test Data			
	Precision	**Recall**	**f1 score**	**Support**	**Precision**	**Recall**	**f1 score**	**Support**
1	0.40	0.50	0.44	4	0.00	0.00	0.00	1
2	1.00	0.67	0.80	3	0.00	0.00	0.00	1
3	0.86	0.86	0.86	14	0.67	1.00	0.80	2
accuracy			0.76	21			0.50	4
macro avg	0.75	0.67	0.70	21	0.22	0.33	0.27	4
weighted avg	0.79	0.76	0.77	21	0.33	0.50	0.40	4

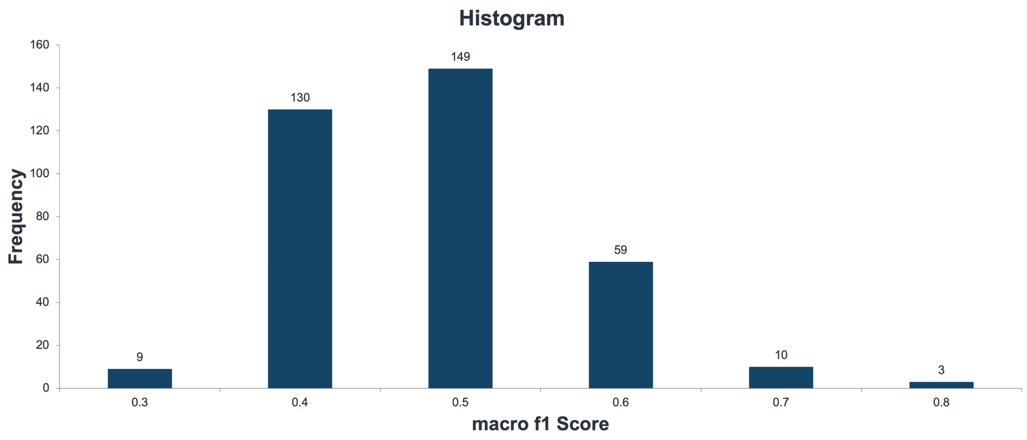


Figure 8. Histogram of *macro f1 scores* obtained on target training data when training 360 times with the same hyperparameters.

8. Discussion

This section discusses the results obtained in the previous sections. In particular, it discusses the individual hyperparameters of the BAR approach and their influence on performance. The second section addresses this paper's limitations and future research directions.

8.1. Interpretation

Evaluating the results of the baseline and BAR approaches shows that both approaches have inherent difficulties in the target domain. In their respective source domains, both methods perform significantly better. The types of difficulties encountered by the approaches differ, however.

The baseline model struggles most with predicting class 2 in the target domain. The other two classes, however, pose relatively few problems for the model.

The BAR approach, on the other hand, handles all classes similarly well. It is better at handling class 2 but tends to perform slightly worse than the baseline approach on classes 1 and 3. However, the BAR approach does not always perform equally well. The training results are subject to strong stochastic fluctuations, so depending on the training, the BAR approach may outperform the baseline approach, or the baseline approach may outperform the BAR approach. Notably, this is true in absolutely identical conditions.

When considering these results, the BAR algorithm's capability to achieve such results without examining the gradient in the learning phase must be considered. This makes the transfer task much harder because the zeroth-order optimization uses only an approximated gradient. This gradient estimation is a significant factor because it enables transfer tasks for complete Black box models with only existing knowledge of in- and outputs. This is an essential benefit of using the BAR approach.

8.2. Hyperparameter Influence

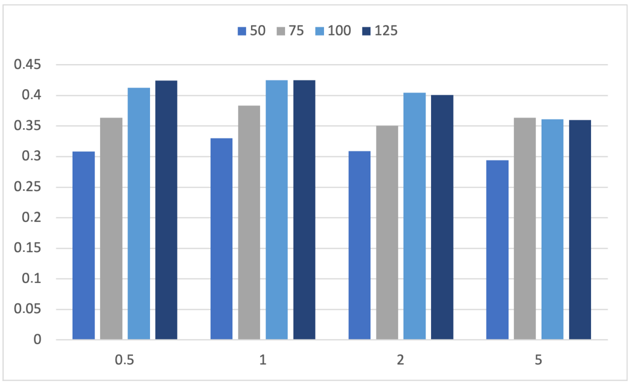
The different hyperparameters influence the performance of an algorithm significantly in different ways. In our case, the (Loss-function) forms the "terrain" on which the zero-order optimization tries to find the parameter setting for the weighting matrix W. Here the weihting factor q of the penalty term influences the structure of this "terrain". The exploration of this "terrain" through the optimization is influenced by the number of randomly generated vectors q and their size *vector_size*. The number can influence the robustness as in all averaging tasks, whereas the size influences the step width in the exploration. Additionally, the learning rate α also influences the exploration pace. To quantitatively investigate the behavior of these hyperparameters and also their interaction, we look at different steps of the gris search and investigate different interactions. These trends are now presented for specific points in the hyperparameter optimization. The following shows each hyperparameter separately and the interaction between the different parameters.

- **Learning Rate α**

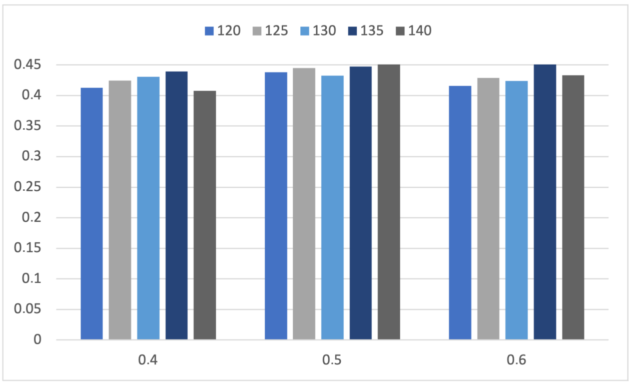
The learning rate is a significant hyperparameter, as in most ML approaches. Because of this, we investigate the effect of different learning rates in combination with the number of random vectors and their size, which can influence the gradient estimation and, therefore, the potential, if or how fast, the solution can converge against a minimum. For this investigation, we look in Figures 9 and 10 at results from the first and third stages of the grid search. In both figures, the x-axis shows different learning rates, while the y-axis is the *macro f1 score*. Figure 9 visualizes with the different bars the influence of the length of a vector (vector size) on the performance. However, besides the vector size, the number of sampled vectors is also performance-critical, especially regarding stochastic behavior, because a larger number of vectors acts through the averaging as a filter. This is visualized in Figure 10 with the different bars for various numbers of vectors in combination with different penalty coefficients and the influence on the variance of the *macro f1 score*.

It comes as no surprise that the learning rate has an impact on the average performance of the algorithm. In the case of this paper, a learning rate between 0.5 and 1 seems to be particularly suitable with respect to the average *macro f1 score* (see Figure 9). For fine-tuning, however, the learning rate seems to play a subordinate role.

In addition, the choice of α affects the variance of the training results. It can be observed that higher learning rates lead to higher variance; see Figure 10. In particular, the interval $[0.5, 1]$ is very sensitive to the variance caused by α.

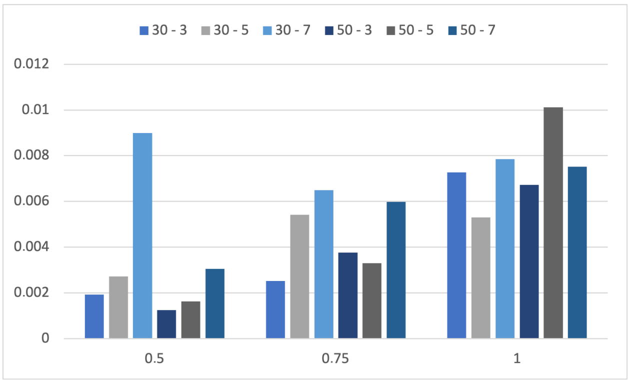


(**a**) First stage of grid search.

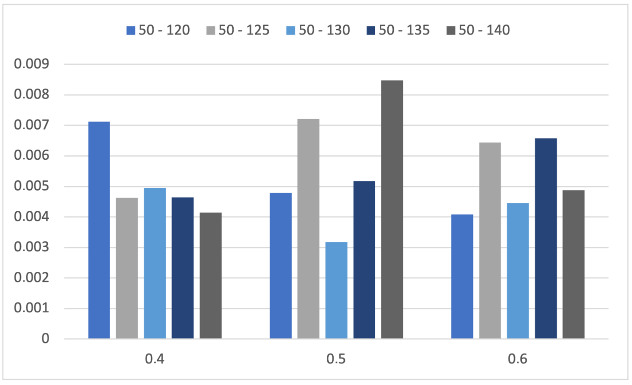


(**b**) Third stage of grid search.

Figure 9. Visual representation of the average *macro f1 scores* over learning rate α (x-axis) and vector size (legend).



(**a**) Second stage of the grid search.



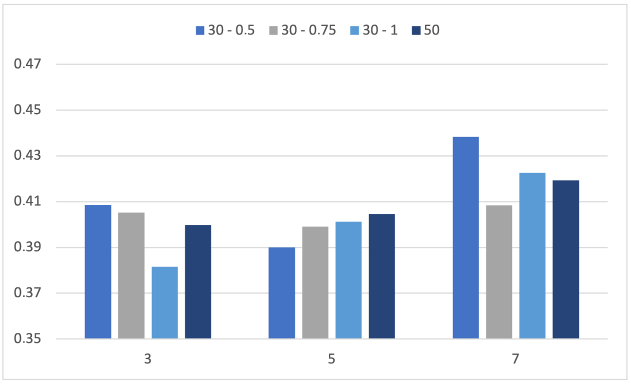
(**b**) Third stage of the grid search.

Figure 10. Visual representation of the performance variance broken down by learning rate α (x-axis) and q and δ (legend).

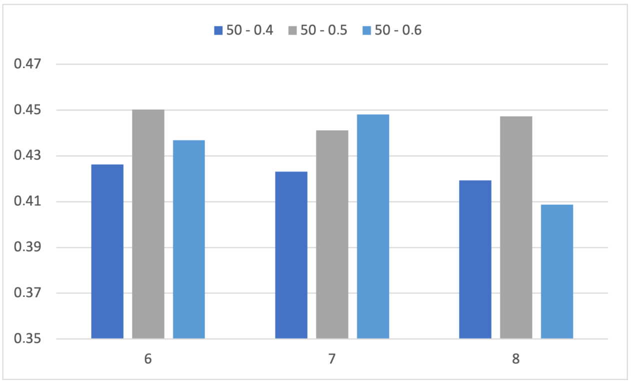
The role of the learning rate can explain these observations. Increasing the learning rate increases the size of the steps in the update function. This allows local minima to be skipped more easily, resulting in better performance. At the same time, the larger steps, combined with the stochastic element in the training, mean that the results can differ more, resulting in a larger variance in performance. A learning rate in the interval $[0.5, 1]$ provides the best performance. However, a comparison of the learning rates 0.4, 0.5, and 0.6 shows that the influence on fine-tuning is marginal. The graphs also show how vector size correlates with performance.

- **Weighting Penalty Term δ**

The Weighting Penalty Term δ is a significant hyperparameter described above because it influences the solution space and gives the approach the possibility to also consider the number of wrong classifications and not only the goal to make predictions with a high probability. Because of this, we investigate the effect of different weighting factors in combination with the number of random vectors and the learning rate. For this investigation, we look in Figure 11 at results from the second and third stages of the grid search. In both figures, the x-axis shows different weighting factors, while the y-axis is the *macro f1 score*. The bars visualize different combinations of the number of generated vectors q with the learning rates α and their influence on the performance.



(**a**) Second stage of the grid search.



(**b**) Third stage of the grid search.

Figure 11. Visual representation of the effect of δ (x-axis) on the average *macro f1 score* of the algorithm, broken down by q and α (legend).

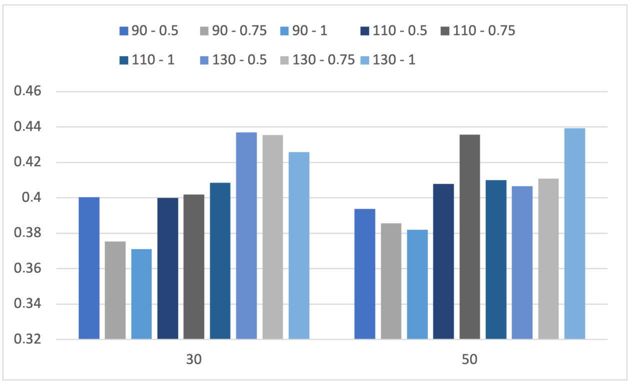
When considering δ, the first thing to note is that $\delta > 0$ must hold for the algorithm to learn the weight vector W. Thus, the penalty term $\mathcal{L}_{PEN}$ is needed in the loss function (15). In addition, it can be seen that the average *macro f1 score* also increases as δ increases; see

Figure 11. However, this is only true to a limited extent. A δ that is too large has a negative effect on the average performance. In this paper, the sweet spot is around $\delta = 7$.

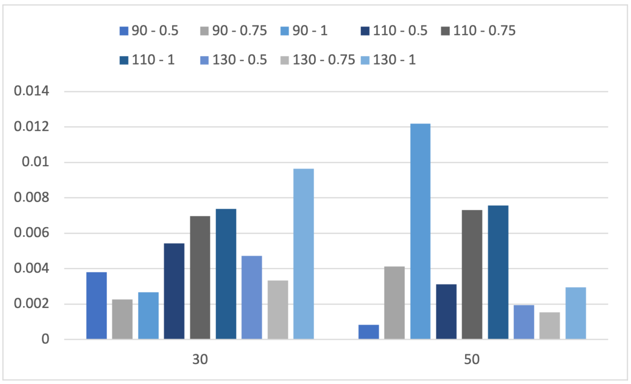
A consistent pattern with respect to the effect of δ on the variance could not be observed.

- **Number Random Vectors q**

The number of random vectors is also of interest, as described above. Because of this, we investigate the effect of a different number of random vectors in combination with the weighting factor and the learning rate. For this investigation, we look in Figure 12 at results from the second stage of the grid search. In the figure, the x-axis shows a different amount of generated vectors, while the y-axis is the *macro f1 score*. The left side visualizes with the different bars the influence of the weighting factor δ and the learning rate α on the performance. As described above, the number of sampled vectors is especially interesting regarding the influence on the stochastic behavior. This is visualized in Figure 12 with the different bars for various penalty coefficients in combination with different learning rates and their influence on the variance in the *macro f1 score*.



(**a**) Second stage of grid search



(**b**) Second stage of grid search

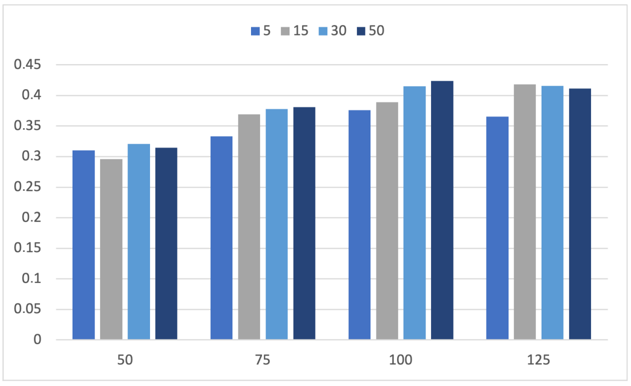
Figure 12. Visual representation of the effect of q (x-axis) on the average *macro f1 score* (**a**) and its variance (**b**), broken down by δ and α (legend).

The number of randomly generated vectors in the training process has a slight influence on the model's performance and variance. A larger number of vectors tends to increase the average performance while decreasing its variance see Figure 12. The extent of the influence on the variance depends, in particular, on the step size of the weight vector updates. Accordingly, the influence of q on the variance increases with increasing vector size or learning rate α. It is reasonable to assume that an increasing number of generated vectors leads to a more accurate estimation of the gradient estimate. As a result, the algorithm will perform better and more consistently.

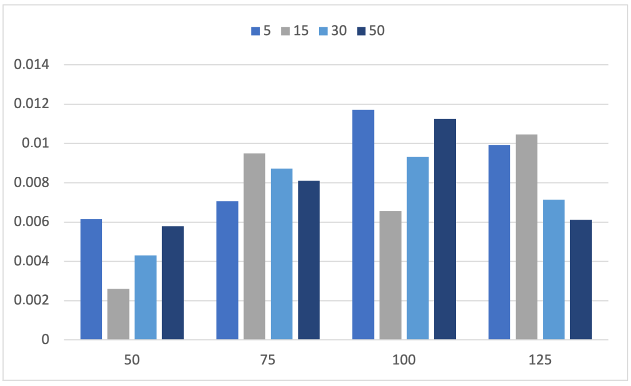
- **Size of Vectors**

Finally, the size of the vectors must be considered. For this purpose, we investigate the effect of different vector sizes in combination with different amounts of random vectors. For this investigation, we look in Figure 13 at results from the second stage of the grid search. In the figure, the x-axis shows a different number of generated vectors, while the y-axis is the *macro f1 score* either absolute (left) or its variance (right). The left side visualizes with the different bars the influence of different numbers of random vectors q on the performance. As described above, the number of sampled vectors is relevant regarding the stochastic behavior. This is visualized on the right side of Figure 13, with different bars for a different amount of generated vectors (as in the left plot) and the corresponding influence on the variance in the *macro f1 score*.

It can be observed that a larger vector size has a positive effect on the average performance of the algorithm (see Figure 13). At the same time, however, a larger vector size is also associated with an increased variance. This finding can also be explained by the fact that a larger vector size is less susceptible to small local minima and thus leads to a better average performance. At the same time, however, the algorithm is less sensitive, which can explain the increased variance.



(**a**) Second stage of grid search



(**b**) Second stage of grid search

Figure 13. Visual representation of the effect of the size of vectors (x-axis) on the average *macro f1 score* (**a**) and its variance (**b**), broken down by q (legend).

9. Limitations and Future Research Directions

A capable black box transfer learning algorithm would offer significant practical benefits. It would enable an end user of various existing models to allow the repurposing of existing models, thereby enhancing versatility and applicability across various domains. This approach would reduce computational costs and training time by leveraging existing models, enabling efficient adaptation to new tasks, particularly in data-limited scenarios. However, there are still limitations to using the BAR algorithm. This section outlines the limitations and future research directions for the Black Box Adversarial Reprogramming (BAR) algorithm, which was developed in this paper. Despite demonstrating potential, the BAR algorithm experiences considerable performance fluctuations due to several constraints, including computational limits and time constraints.

Key limitations include the small size of the target dataset, which comprises only 25 instances, suggesting that a larger dataset might improve algorithm performance. Additionally, the disparity in feature counts between source and target domains raises questions about optimal feature ratios for model training. Explorations into feature engineering and possibly using principal component analysis (PCA) to adjust feature counts in different domains are recommended.

The research also notes that the current hyperparameter tuning process, a grid search, is limited by time and could benefit from a more extensive exploration to possibly reduce performance variance. Addressing the domain gap between data types (e.g., turbofan engine and rolling bearing data) used in baseline and BAR approaches may also provide insights into enhancing model adaptability and performance.

Future directions include testing different optimization strategies beyond the zeroth-order, the greedy approach currently employed, which may lead to suboptimal solutions. Investigating alternative loss functions and methods for initializing weight vectors could refine the training process and achieve more consistent results. Moreover, exploring different methods for output translation in the adversarial reprogramming context, such as varying aggregation strategies, may offer further improvements in model performance.

Author Contributions: Conceptualization, A.B.; methodology, A.B. and F.S.; software, F.S.; validation, F.S.; formal analysis, A.B. and F.S.; investigation, A.B. and F.S.; resources, J.F.; data curation, F.S.; writing—original draft preparation, A.B. and F.S.; writing—review and editing, A.B. and A.P.; visualization, F.S.; supervision, A.P. and J.F.; project administration, A.B.; funding acquisition, A.P. and J.F. All authors have read and agreed to the published version of the manuscript.

Funding: We acknowledge support by the KIT-Publication Fund of the Karlsruhe Institute of Technology.

Data Availability Statement: For the study, the publicly available PRONOSTIA and CMAPSS datasets were used.

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

References

1. Maslej, N.; Fattorini, L.; Brynjolfsson, E.; Etchemendy, J.; Ligett, K.; Lyons, T.; Manyika, J.; Ngo, H.; Niebles, J.C.; Sellitto, M.; et al. The AI Index 2023 Annual Report. *arXiv* **2023**, arXiv.2310.03715.
2. Petangoda, J.; Deisenroth, M.P.; Monk, N.A. Learning to Transfer: A Foliated Theory. *arXiv* **2021**, arXiv:2107.10763.
3. Elsayed, G.F.; Goodfellow, I.J.; Sohl-Dickstein, J.N. Adversarial Reprogramming of Neural Networks. *arXiv* **2018**, arXiv:1806.11146.
4. Tsai, Y.Y.; Chen, P.Y.; Ho, T.Y. Transfer Learning without Knowing: Reprogramming Black-box Machine Learning Models with Scarce Data and Limited Resources. *arXiv* **2020**, arXiv:2007.08714.
5. Nectoux, P.; Gouriveau, R.; Medjaher, K.; Ramasso, E.; Chebel-Morello, B.; Zerhouni, N.; Varnier, C. PRONOSTIA: An experimental platform for bearings accelerated degradation tests. In Proceedings of the IEEE International Conference on Prognostics and Health Management (PHM'12), Beijing, China, 23–25 May 2012; IEEE Catalog Number: CPF12PHM-CDR; pp. 1–8.
6. Bengio, Y.; Goodfellow, I.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2017; Volume 1.
7. Pan, S.J.; Yang, Q. A Survey on Transfer Learning. *IEEE Trans. Knowl. Data Eng.* **2010**, *22*, 1345–1359. [CrossRef]
8. Zhuang, F.; Qi, Z.; Duan, K.; Xi, D.; Zhu, Y.; Zhu, H.; Xiong, H.; He, Q. A Comprehensive Survey on Transfer Learning. *Proc. IEEE* **2019**, *109*, 43–76. [CrossRef]
9. Azari, M.S.; Flammini, F.; Santini, S.; Caporuscio, M. A Systematic Literature Review on Transfer Learning for Predictive Maintenance in Industry 4.0. *IEEE Access* **2023**, *11*, 12887–12910. [CrossRef]
10. Dai, W.; Yang, Q.; Xue, G.R.; Yu, Y. Boosting for transfer learning. In Proceedings of the 24th International Conference on Machine Learning, Corvallis, OR, USA, 20–24 June 2007; pp. 193–200.
11. Yao, Y.; Doretto, G. Boosting for transfer learning with multiple sources. In Proceedings of the 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Francisco, CA, USA, 13–18 June 2010; pp. 1855–1862.
12. Huang, J.; Gretton, A.; Borgwardt, K.; Schölkopf, B.; Smola, A. Correcting sample selection bias by unlabeled data. In Proceedings of the Advances in Neural Information Processing Systems 19 (NIPS 2006), Vancouver, BC, Canada, 4–7 December 2006; Volume 19.
13. Jiang, J.; Zhai, C. Instance weighting for domain adaptation in NLP. In Proceedings of the Annual Meeting of the Association for Computational Linguistics, Prague, Czech Republic, 23–30 June 2007; ACL: Philadelphia, PA, USA, 2007.
14. Dai, W.; Xue, G.R.; Yang, Q.; Yu, Y. Transferring naive bayes classifiers for text classification. In Proceedings of the AAAI, Vancouver, BC, Canada, 22–26 July 2007; Volume 7, pp. 540–545.
15. Asgarian, A.; Sobhani, P.; Zhang, J.C.; Mihailescu, M.; Sibilia, A.; Ashraf, A.B.; Taati, B. A hybrid instance-based transfer learning method. *arXiv* **2018**, arXiv:1812.01063.
16. Chen, Q.; Xue, B.; Zhang, M. Instance based transfer learning for genetic programming for symbolic regression. In Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, 10–13 June 2019; pp. 3006–3013.
17. Raina, R.; Battle, A.; Lee, H.; Packer, B.; Ng, A.Y. Self-taught learning: Transfer learning from unlabeled data. In Proceedings of the 24th International Conference on Machine Learning, Corvallis, OR, USA, 20–24 June 2007; pp. 759–766.
18. Daumé III, H. Frustratingly easy domain adaptation. *arXiv* **2009**, arXiv:0907.1815.
19. Yan, H.; Ding, Y.; Li, P.; Wang, Q.; Xu, Y.; Zuo, W. Mind the class weight bias: Weighted maximum mean discrepancy for unsupervised domain adaptation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 2272–2281.
20. Blitzer, J.; McDonald, R.; Pereira, F. Domain adaptation with structural correspondence learning. In Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing, Sydney, Australia, 22–23 July 2006; pp. 120–128.
21. Argyriou, A.; Evgeniou, T.; Pontil, M. Multi-task feature learning. In Proceedings of the Advances in Neural Information Processing Systems 19 (NIPS 2006), Vancouver, BC, Canada, 4–6 December 2006; Volume 19.
22. Argyriou, A.; Pontil, M.; Ying, Y.; Micchelli, C. A spectral regularization framework for multi-task structure learning. In Proceedings of the Advances in Neural Information Processing Systems 20 (NIPS 2007), Vancouver, BC, Canada, 3–6 December 2007; Volume 20.
23. Dai, W.; Xue, G.R.; Yang, Q.; Yu, Y. Co-clustering based classification for out-of-domain documents. In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, CA, USA, 12–15 August 2007; pp. 210–219.
24. Johnson, R.; Zhang, T. A high-performance semi-supervised learning method for text chunking. In Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05), Ann Arbor, MI, USA, 25–30 June 2005; pp. 1–9.
25. Bonilla, E.V.; Chai, K.; Williams, C. Multi-task Gaussian process prediction. In Proceedings of the Advances in Neural Information Processing Systems 20 (NIPS 2007), Vancouver, BC, Canada, 3–6 December 2007; Volume 20.

26. Lawrence, N.D.; Platt, J.C. Learning to learn with the informative vector machine. In Proceedings of the Twenty-First International Conference on Machine Learning, Banff, AB, Canada, 4–8 July 2004; p. 65.
27. Evgeniou, T.; Pontil, M. Regularized multi–task learning. In Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, WA, USA, 22–25 August 2004; pp. 109–117.
28. Duan, L.; Tsang, I.W.; Xu, D.; Chua, T.S. Domain adaptation from multiple sources via auxiliary classifiers. In Proceedings of the 26th Annual International Conference on Machine Learning, Montreal, QC, Canada, 14–18 June 2009; pp. 289–296.
29. Duan, L.; Xu, D.; Tsang, I.W.H. Domain adaptation from multiple sources: A domain-dependent regularization approach. *IEEE Trans. Neural Netw. Learn. Syst.* **2012**, *23*, 504–518. [CrossRef]
30. Blum, A.; Mitchell, T. Combining labeled and unlabeled data with co-training. In Proceedings of the Eleventh Annual Conference on Computational Learning Theory, Madison, WI, USA, 24–26 July 1998; pp. 92–100.
31. Zhuang, F.; Luo, P.; Xiong, H.; He, Q.; Xiong, Y.; Shi, Z. Exploiting associations between word clusters and document classes for cross-domain text categorization. *Stat. Anal. Data Min. Asa Data Sci. J.* **2011**, *4*, 100–114. [CrossRef]
32. Oquab, M.; Bottou, L.; Laptev, I.; Sivic, J. Learning and transferring mid-level image representations using convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; pp. 1717–1724.
33. Huang, J.T.; Li, J.; Yu, D.; Deng, L.; Gong, Y. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 26–31 May 2013; pp. 7304–7308.
34. Long, M.; Zhu, H.; Wang, J.; Jordan, M.I. Unsupervised domain adaptation with residual transfer networks. In Proceedings of the Advances in Neural Information Processing Systems 29 (NIPS 2016), Barcelona, Spain, 5–10 December 2016; Volume 29.
35. George, D.; Shen, H.; Huerta, E. Deep Transfer Learning: A new deep learning glitch classification method for advanced LIGO. *arXiv* **2017**, arXiv:1706.07446.
36. Mihalkova, L.; Huynh, T.; Mooney, R.J. Mapping and revising markov logic networks for transfer learning. In Proceedings of the AAAI, Vancouver, BC, Canada, 22–26 July 2007; Volume 7, pp. 608–614.
37. Mihalkova, L.; Mooney, R.J. Transfer learning by mapping with minimal target data. In Proceedings of the AAAI-08 Workshop on Transfer Learning for Complex Tasks, Chicago, IL, USA, 13–14 July 2008; pp. 31–36.
38. Davis, J.; Domingos, P. Deep transfer via second-order markov logic. In Proceedings of the 26th Annual International Conference on Machine Learning, Montreal, QC, Canada, 14–18 June 2009; pp. 217–224.
39. Zonta, T.; Costa, C.A.d.; Righi, R.d.R.; Lima, M.J.; Trindade, E.S.d.; Li, G.P. Predictive maintenance in the Industry 4.0: A systematic literature review. *Comput. Ind. Eng.* **2020**, *150*, 106889. [CrossRef]
40. Ran, Y.; Zhou, X.; Lin, P.; Wen, Y.; Deng, R. A survey of predictive maintenance: Systems, purposes and approaches. *arXiv* **2019**, arXiv:1912.07383.
41. Jimenez, J.J.M.; Schwartz, S.; Vingerhoeds, R.; Grabot, B.; Salaün, M. Towards multi-model approaches to predictive maintenance: A systematic literature survey on diagnostics and prognostics. *J. Manuf. Syst.* **2020**, *56*, 539–557. [CrossRef]
42. Zheng, H.; Wang, R.; Yang, Y.; Yin, J.; Li, Y.; Li, Y.; Xu, M. Cross-Domain Fault Diagnosis Using Knowledge Transfer Strategy: A Review. *IEEE Access* **2019**, *7*, 129260–129290. [CrossRef]
43. Mao, W.; He, J.; Zuo, M.J. Predicting Remaining Useful Life of rolling bearings based on deep feature representation and transfer learning. *IEEE Trans. Instrum. Meas.* **2019**, *69*, 1594–1608. [CrossRef]
44. Wen, L.; Gao, L.; Li, X. A New Deep Transfer Learning Based on Sparse Auto-Encoder for Fault Diagnosis. *IEEE Trans. Syst. Man Cybern. Syst.* **2019**, *49*, 136–144. [CrossRef]
45. Zhu, J.; Chen, N.; Shen, C. A new data-driven transferable Remaining Useful Life prediction approach for bearing under different working conditions. *Mech. Syst. Signal Process.* **2020**, *139*, 106602. [CrossRef]
46. da Costa, P.R.d.O.; Akçay, A.; Zhang, Y.; Kaymak, U. Remaining Useful Lifetime prediction via deep domain adaptation. *Reliab. Eng. Syst. Saf.* **2020**, *195*, 106682. [CrossRef]
47. Xia, P.; Huang, Y.; Li, P.; Liu, C.; Shi, L. Fault knowledge transfer assisted ensemble method for remaining useful life prediction. *IEEE Trans. Ind. Inform.* **2021**, *18*, 1758–1769. [CrossRef]
48. Ma, X.; Niu, T.; Liu, X.; Luan, H.; Zhao, S. Remaining Useful Lifetime prediction of rolling bearing based on ConvNext and multi-feature fusion. In Proceedings of the 2022 International Conference on Computer Engineering and Artificial Intelligence (ICCEAI), Shijiazhuang, China, 22–24 July 2022; pp. 299–304.
49. Xu, G.; Liu, M.; Jiang, Z.; Shen, W.; Huang, C. Online Fault Diagnosis Method Based on Transfer Convolutional Neural Networks. *IEEE Trans. Instrum. Meas.* **2020**, *69*, 509–520. [CrossRef]
50. Cheng, H.; Kong, X.; Wang, Q.; Ma, H.; Yang, S.; Chen, G. Deep transfer learning based on dynamic domain adaptation for Remaining Useful Life prediction under different working conditions. *J. Intell. Manuf.* **2023**, *34*, 587–613. [CrossRef]
51. Ong, K.S.H.; Wang, W.; Hieu, N.Q.; Niyato, D.T.; Friedrichs, T. Predictive Maintenance Model for IIoT-Based Manufacturing: A Transferable Deep Reinforcement Learning Approach. *IEEE Internet Things J.* **2022**, *9*, 15725–15741. [CrossRef]
52. Xu, Y.; Sun, Y.; Liu, X.; Zheng, Y. A digital-twin-assisted fault diagnosis using deep transfer learning. *IEEE Access* **2019**, *7*, 19990–19999. [CrossRef]
53. Kim, H.; Youn, B.D. A new parameter repurposing method for parameter transfer with small dataset and its application in fault diagnosis of rolling element bearings. *IEEE Access* **2019**, *7*, 46917–46930. [CrossRef]

54. Shao, S.; McAleer, S.; Yan, R.; Baldi, P. Highly Accurate Machine Fault Diagnosis Using Deep Transfer Learning. *IEEE Trans. Ind. Inform.* **2019**, *15*, 2446–2455. [CrossRef]
55. Samek, W.; Wiegand, T.; Müller, K.R. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv* **2017**, arXiv:1708.08296.
56. Xu, G.; Liu, M.; Wang, J.; Ma, Y.; Wang, J.; Li, F.; Shen, W. Data-driven fault diagnostics and prognostics for predictive maintenance: A brief overview. In Proceedings of the 2019 IEEE 15th International Conference On Automation Science and Engineering (CASE), Vancouver, BC, Canada, 22–26 August 2019; pp. 103–108. [CrossRef]
57. Arrieta, A.B.; Díaz-Rodríguez, N.; Del Ser, J.; Bennetot, A.; Tabik, S.; Barbado, A.; García, S.; Gil-López, S.; Molina, D.; Benjamins, R. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Inf. Fusion* **2020**, *58*, 82–115. [CrossRef]
58. Mahyari, A.G.; Locher, T. Robust predictive maintenance for robotics via unsupervised transfer learning. In Proceedings of the International FLAIRS Conference Proceedings, North Miami Beach, FL, USA, 17–19 May 2021; p. 34. [CrossRef]
59. Mao, W.; He, J.; Sun, B.; Wang, L. Prediction of bearings Remaining Useful Life across working conditions based on transfer learning and time series clustering. *IEEE Access* **2021**, *9*, 135285–135303. [CrossRef]
60. Shen, F.; Yan, R. A new intermediate-domain SVM-based transfer model for rolling bearing RUL prediction. *IEEE/ASME Trans. Mechatron.* **2021**, *27*, 1357–1369. [CrossRef]
61. Schlag, S.; Schmitt, M.; Schulz, C. Faster support vector machines. *J. Exp. Algorithmics (JEA)* **2021**, *26*, 1–21. [CrossRef]
62. Wang, X.; Zhang, H.H.; Wu, Y. Multiclass probability estimation with support vector machines. *J. Comput. Graph. Stat.* **2019**, *28*, 586–595. [CrossRef]
63. Olson, M.A.; Wyner, A.J. Making sense of random forest probabilities: A kernel perspective. *arXiv* **2018**, arXiv:1812.05792.
64. Mathew, V.; Toby, T.; Singh, V.; Rao, B.M.; Kumar, M.G. Prediction of Remaining Useful Lifetime (RUL) of turbofan engine using machine learning. In Proceedings of the 2017 IEEE International Conference on Circuits and Systems (ICCS), Thiruvananthapuram, India, 20–21 December 2017; pp. 306–311.
65. Saxena, A.; Goebel, K.; Simon, D.; Eklund, N. Damage propagation modeling for aircraft engine run-to-failure simulation. In Proceedings of the 2008 International Conference on Prognostics and Health Management, Denver, CO, USA, 6–9 October 2008; pp. 1–9.
66. Wu, Z.; Yu, S.; Zhu, X.; Ji, Y.; Pecht, M. A weighted deep domain adaptation method for industrial fault prognostics according to prior distribution of complex working conditions. *IEEE Access* **2019**, *7*, 139802–139814. [CrossRef]
67. Xia, M.; Li, T.; Shu, T.; Wan, J.; De Silva, C.W.; Wang, Z. A two-stage approach for the Remaining Useful Life prediction of bearings using deep neural networks. *IEEE Trans. Ind. Inform.* **2018**, *15*, 3703–3711. [CrossRef]
68. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
69. Prinzie, A.; Van den Poel, D. Random forests for multiclass classification: Random multinomial logit. *Expert Syst. Appl.* **2008**, *34*, 1721–1732. [CrossRef]
70. Krmar, J.; Vukićević, M.; Kovačević, A.; Protić, A.; Zečević, M.; Otašević, B. Performance comparison of nonlinear and linear regression algorithms coupled with different attribute selection methods for quantitative structure-retention relationships modelling in micellar liquid chromatography. *J. Chromatogr. A* **2020**, *1623*, 461146. [CrossRef] [PubMed]
71. Song, Y.Y.; Ying, L. Decision tree methods: Applications for classification and prediction. *Shanghai Arch. Psychiatry* **2015**, *27*, 130.
72. de Mathelin, A.; Deheeger, F.; Richard, G.; Mougeot, M.; Vayatis, N. Adapt: Awesome domain adaptation python toolbox. *arXiv* **2021**, arXiv:2107.03049.
73. Lemaître, G.; Nogueira, F.; Aridas, C.K. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *J. Mach. Learn. Res.* **2017**, *18*, 559–563.