Human-in-the-Loop Large-Scale Predictive Maintenance of Workstations

Alexander Nikitin
alexander.nikitin@aalto.fi
Department of Computer Science, Aalto University
Espoo, Finland

Samuel Kaski
samuel.kaski@aalto.fi
Department of Computer Science, Aalto University
Espoo, Finland
Department of Computer Science, University of Manchester
Manchester, UK

ABSTRACT

Predictive maintenance (PdM) is the task of scheduling maintenance operations based on a statistical analysis of the system's condition. We propose a human-in-the-loop PdM approach in which a machine learning system predicts future problems in sets of workstations (computers, laptops, and servers). Our system interacts with domain experts to improve predictions and elicit their knowledge. In our approach, domain experts are included in the loop not only as providers of correct labels, as in traditional active learning, but as a source of explicit decision rule feedback. The system is automated and designed to be easily extended to novel domains, such as maintaining workstations of several organizations. In addition, we develop a simulator for reproducible experiments in a controlled environment and deploy the system in a large-scale case of real-life workstations PdM with thousands of workstations for dozens of companies.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Applied computing**;

KEYWORDS

machine learning, human-in-the-loop, predictive maintenance, Bayesian optimization, applications

ACM Reference Format:
Alexander Nikitin and Samuel Kaski. 2022. Human-in-the-Loop Large-Scale Predictive Maintenance of Workstations. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3534678.3539196

1 INTRODUCTION

Predictive maintenance (PdM) determines the optimal timepoint for maintenance actions based on the condition of equipment. This task is becoming one of the most critical problems in systems management. For example, within the Industry 4.0 concept [7], PdM is described as an essential part of computer-assisted manufacturing. PdM techniques have been applied in several domains, including manufacturing [17], the Internet of Things [8], and air force systems [15].

The development of effective PdM systems for workstations is driven by the growth in the number of personal computers, laptops, and servers. In this application, PdM systems will enhance the efficiency of maintenance engineers and improve customer experience. However, while predictive maintenance approaches have been used in many fields (see Sec. 2 for a more detailed discussion), to the best of our knowledge, no studies have proposed and field-tested machine learning (ML) solutions to the PdM of workstations.

We study the predictive maintenance task through the lens of human-in-the-loop (HITL) machine learning methods. HITL ML is a set of ML techniques that interact with human experts in the decision-making loop. HITL methods produce various benefits, including better predictive performance [24] and domain adaptation [25]. These approaches are especially beneficial for safety-critical tasks, such as airborne collision avoidance [22] and surgery automation [12]. The success of prior works in applying HITL approaches to critical tasks and the ability of HITL to assist in decision-making indicate the potential for employing these methods in PdM.

Contributions. In this paper, we: *(i)* formulate the predictive maintenance problem for workstations, *(ii)* propose ML methods for workstation PdM, *(iii)* explain how domain experts can be effectively included in the PdM process using HITL machine learning approaches, and *(iv)* describe a field-tested large-scale implementation of our approach and motivate the system design choices.

Reproducibility. We validate the developed system using both offline and online. For offline validation, we measure the model's performance on historical data and develop a simulator. The simulator allows for generating datasets that mimic properties of the real data and allows for testing the methods in a controllable environment where the generation process parameters can be varied. The source code of the toy experiments with synthetic data and the simulator are available at https://github.com/AaltoPML/human-in-the-loop-predictive-maintenance. Also, we describe the experiments with historical data collected from one thousand workstations to validate the models and online validation of the system in the field.

Application benefits. This work is important for several parties: maintenance engineers/domain experts, companies with extensive IT infrastructure, and machine learning researchers. For domain

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539196

experts, the work proposes a tool that allows them not only to prevent upcoming malfunction and automatize routine parts of their work but also provides a user-friendly, interactive, and reliable assistant. For businesses, this tool allows cost-efficient workstation monitoring. It consequently leads to fixing possible malfunctions in advance, without hurting the user experience. Lastly, for machine learning researchers, this application shows an example of effective interaction with humans in PdM applications and, importantly, a reproducible way to simulate and test their methods on synthetic PdM data.

2 BACKGROUND

Predictive maintenance. Machine learning approaches to PdM include a variety of methods ranging from knowledge-based techniques to deep learning (DL) systems [27]. Recent developments have primarily focused on DL approaches, including autoencoders [28], convolutional neural networks [33], and recurrent neural networks [18]. Knowledge-based systems were popular in the early days of machine learning, but, currently, they have given way to data-driven approaches. In terms of applications, predictive maintenance methods have often been applied to high-cost systems, including ships, aircraft fleets, and turbofans [27], [29]. However, with the increase of data available for more standard systems, the scope is widening to encompass more common problems, such as those related to the Internet of Things [4], and ATMs [31]. Several approaches to the analysis of the tickets in IT services were developed, for example, [35] and [26]. These approaches focus on extracting useful information from the textual description and otherwise improve reaction time; they do not tackle the proactive incident management aspect, which is the main focus of our work. Scalability and deployment of the methods in changing environments remain open challenges for all application areas in predictive maintenance, and we resolve parts of them in this work.

Human-in-the-loop machine learning. Machine learning agents can gain useful information from interactions with humans (HITL methods). These methods have resulted in many advantages, including performance improvements [1], better explainability [5], and tackling computationally hard problems [20]. Humans may assist the learning process in various ways, for example, by including inductive biases in building kernel machines [32]. In this work, we show how ML models can interact with experts in the PdM process via explicit decision rule feedback.

Recommender systems. Recommender systems are machine learning methods used to predict the relevancy of a given collection of items to a group of users. Recommender systems have been applied in many other domains, including music [34], and research articles [2]. Nevertheless, they have been applied to PdM only seldom. For example, in [10], the authors showed how to recommend maintenance actions for an industrial piece of equipment. We utilize a recommender engine with additional information to represent the models' results.

3 PREDICTIVE MAINTENANCE OF WORKSTATIONS

We consider the task of PdM of a set of workstations: personal laptops, servers, and other computers. Our goal is to develop a system that will predict whether a problematic situation will occur in the future and guide domain experts to prevent it. An essential practical requirement is that the system should be applicable to different organizations' workstations (in ML terms, automatically adapt to novel domains). Specifically, we need to deal with dozens of organizations with thousands of workstations each. The development of the machine learning system consists of the following steps, described in more detail in Sec. 5: extracting numerical measurements from the workstations, processing the extracted features and preparing an ML dataset, application of ML algorithm to the extracted dataset, deployment and continuous integration of the model in the PdM pipeline, and collecting feedback from domain experts. Importantly, we allow domain experts to give explicit decision rule feedback, which improves the model's predictive performance and allows experts to influence the system's behavior.

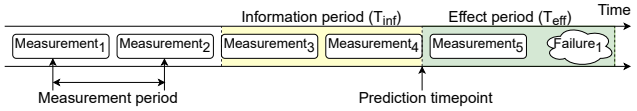


Figure 1: The system analyzes aggregated measurements and predicts future (relative to prediction timepoint) failures. The information period is a time segment during which the data is aggregated. The effect period is a period in the future relative to the prediction timepoint, which is used to calculate the target variable (an indicator of a failure).

3.1 Dataset

Data collection can be performed using agents installed on each maintained computer. The agents submit their measurements once every measurement period (here two hours), resulting in a heterogeneous set of temporal measurements. For learning to predict failures in the future, we define two time intervals: *information period* (T_{inf}) and *effect period* (T_{eff}). Let us consider an arbitrary time point and an arbitrary workstation. We analyze a period before the prediction point (information period) to gather workstation measurements. Then, we make predictions of breakdowns in a specific time interval in the future (effect period). Schematically, the information and effect periods are shown in Fig. 1. To construct an ML task out of these data, we need to specify a method for generating covariates (inputs to the predictor) and target variables.

Covariates. We use several informative values that describe the condition of the workstations; these measurements include CPU load, used memory, installed drivers, and software monitoring alarms. The features of a particular workstation can be described as a sequence of data points $\boldsymbol{x} \in \mathbb{R}^d$ for each timestamp, where d is the number of numerical measurements that characterize the workstations. The dataset $\boldsymbol{X}$ is a collection of situations where each element $\boldsymbol{x_i} \in \boldsymbol{X}$ is a set of collected measurements over T_{inf} periods. The data points are either aggregated over each period to multi-dimensional vectors $\boldsymbol{x_i} \in \mathbb{R}^d$ as described in Eq. 1 or to time-indexed measurements $\boldsymbol{x_i} \in \mathbb{R} \times \mathbb{R}^d$. We aggregate the measurements over the information periods and extract statistics that

we use as features, using the following approach:

$$\tilde{\boldsymbol{x}}_i = \bigoplus_{\text{agg}\in\mathcal{A}} \text{agg}\left(\boldsymbol{x}_i^{t-T_{\text{inf}}}, \ldots, \boldsymbol{x}_i^{t}\right), \tag{1}$$

where $\mathcal{A}$ is a set of the aggregational operators, and $\oplus$ concatenates results into one vector. The aggregational operators can include, for example, percentiles, rolling mean, or mode. Later in the text, we will denote aggregated features as $\boldsymbol{x}_i$ and $\boldsymbol{X}$.

Some of the extracted information can be represented as sets; for example, the sets of installed drivers or software programs. We have applied a natural language processing (NLP) technique called tf-idf [21] to transform temporal information about sets into covariates. Consider installed drivers as an example. We transformed the collected data on drivers to a list of pairs (name and version) and computed the standard measures term frequency (tf) and inverse document frequency (idf) as follows:

$$\begin{aligned} \text{tf}(d, \boldsymbol{x}) &= \frac{\#\{d' = d | d' \in \boldsymbol{x}\}}{\#\{d' = d | d' \in \boldsymbol{x}', \boldsymbol{x}' \in \boldsymbol{X}\}}, \\ \text{idf}(d) &= \log \frac{N}{\#\{d \in \boldsymbol{x} | \boldsymbol{x} \in \boldsymbol{X}\}}, \\ \text{tf-idf}(d, \boldsymbol{x}) &= \text{tf}(d, \boldsymbol{x}) \times \text{idf}(d), \end{aligned} \tag{2}$$

where d is a driver, $\boldsymbol{x}$ is an element of the training set, and D is a whole set of drivers. With symbol #, we denote the cardinality of a set, and N is the size of the training dataset. We use the notation $d \in \boldsymbol{x}$ to denote that the driver d was installed at some point in the x's information period. The tf-idf representation of installed drivers is then concatenated to the vector with the rest of the features.

Targets. The machine learning model M tries to predict whether a problem will occur in the following T_{eff} relative to the prediction point. To indicate the health of a particular workstation, we count system errors and warnings (in this paper, we will call them alerts) for each workstation during the effect period. For example, some alerts indicate a problem loading the system (errors for slow startup) or report software's frequent crashes. To distinguish failures from normal functioning, we compare the total number of alerts with a percentage threshold, which results in a binary classification problem. From discussions with domain experts, we noticed that they considered the total number of errors and alerts to be a more fine-grained indication of problems than other approaches (e.g., counting unique types of alerts or user requests).

3.2 Bayesian optimization of T_{inf} and T_{eff}

In order to transform raw data into a machine learning dataset, it is crucial to select optimal information and effect periods. Let us consider function perf($T_{\text{inf}}, T_{\text{eff}}$, M, D), which approximates the predictive performance of model M on dataset D with T_{inf} and T_{eff} parameters. To find the optimal periods, we might take the brute-force grid search approach and evaluate perf for a large set of values. That would, however, be very inefficient as we need to employ the system for different organizations and repeat this process. Moreover, to get statistically significant estimates of perf, we have to run computationally inefficient cross-validation for each combination of the parameters. Rather than adapting the brute-force approach, we optimize the function perf with Bayesian optimization (BO) [23].

To employ the BO methods, it is necessary to define a surrogate function that will approximate the model's generalization performance. For this purpose, we define a Gaussian process (GP) prior over functions:

$$f \sim \mathcal{GP}(\mu(\mathbf{x}), K(\mathbf{x}, \mathbf{x})). \tag{3}$$

GPs are described with a mean function $\mu(\boldsymbol{x})$ and a covariance function (kernel) K, such as squared exponential, periodic, and radial basis function (RBF) kernels [11]. In our work, we used the RBF kernel:

$$K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \left\|\mathbf{x} - \mathbf{x}'\right\|^2), \tag{4}$$

where γ is a length scale hyperparameter. For each iteration, the BO algorithm selects the next most informative point and evaluates the function at this point. This selection process is organised by the function called acquisition function. In our work, we used the upper confidence bound (UCB) acquisition function [3] which allows the algorithm to balance between exploration and exploitation steps:

$$a_{UCB} = m(\mathbf{x}) + \kappa\sigma(\mathbf{x}), \tag{5}$$

where κ is the exploration-exploitation tradeoff parameter and $\sigma(\boldsymbol{x}) = \sqrt{K(\boldsymbol{x}, \boldsymbol{x})}$. After choosing the point, the BO algorithm updates the GP posterior with the acquired value.

4 HUMAN-IN-THE-LOOP PREDICTIVE MAINTENANCE

4.1 Base ML model

We employ a machine learning algorithm to predict whether a problem will occur in the following T_{eff} time. More formally, our aim is to develop a binary classifier $\mathbf{M} : \boldsymbol{X} \rightarrow \{0, 1\}$, where each element of $\boldsymbol{X}$ is an aggregated set of workstation measurements during T_{inf} (yellow segment in Fig. 1), and 0 indicates normal functioning of a system and 1 malfunctioning.

As a base ML model, we consider several approaches that have shown excellent results in ML problems with extensively heterogeneous data: gradient boosting [13], and extremely randomized trees [14]. We also compared the results to logistic regression. These methods possess several advantages over deep learning approaches, such as long short-term memory (LSTM) [19], and gated recurrent unit (GRU) [6] neural networks. These advantages include the ability to cope with a large number of noisy and missing measurements and interpretability, the latter being a requirement for the machine learning method in our solution and being also important for other critical systems.

4.2 Decision rule elicitation (DRE)

In this work, we propose to include the domain experts in the predictive maintenance loop in a novel way for PdM — via explicit decision rule elicitation [25]. Decision rule elicitation provides several advantages over standard techniques: better domain adaptation, predictive performance, and explainability.

An informal survey among maintenance engineers showed that they were able to explicitly explain their decision-making process, justifying what heuristics they used to check whether a workstation was broken. Such explicit heuristics can be represented as decision trees or logical formulas, in which the experts often rely on the current warnings and alarms and compare measurements (e.g., hard

disk free space) with some threshold values they have identified based on their earlier experience. An example of a decision rule is

$$\begin{cases} 1, \text{if } c((a_{\text{perf}}, a_{\text{mem}}), \boldsymbol{x}) > 0 \vee c(a_{\text{soft}}, \boldsymbol{x}) > 1, \\ 0.5, \text{otherwise}, \end{cases} \tag{6}$$

where a_* are alerts related to different aspects of the workstation's operation, in this example, performance, memory, and software, x is an observation, and c is a function that returns the number of alerts during the information period of the observation. The value one in the first equation means that the device has to be examined immediately, and 0.5 in the second means that there is not enough information to determine the workstation condition. These values can be interpreted as a subjective probability of malfunctioning.

Following the approach proposed in [25], we query experts for decision rules when they discover a particular prediction to be wrong. The decision rule feedback is taken into account with the following model: Consider the explicit decision feedback rules from the user as a set of Boolean predicates, $f_1, f_2, \ldots, f_n$ elicited from domain experts. We generalize this approach to return a discrete probability distribution over the target variable domain: $f : \boldsymbol{X} \longrightarrow \mathbf{P}$. In our case, $\mathbf{P}$ refers to the probability of a malfunction. The feedback rules were provided by domain experts in natural language and then manually translated into the set of functions. The resulting classifier can be written as:

$$C_{\text{feedback}}(\boldsymbol{x}) = \zeta\Big(\sum_{i=1}^{F} f_i(\boldsymbol{x})\, \text{sim}(\boldsymbol{X}^i_{\text{test}}, \boldsymbol{x})\, \theta_i\Big). \tag{7}$$

Here F is the total number of feedback rules, θ are weights of the rules that could be either learned from the data or manually selected using prior knowledge, sim is a similarity measure that is calculated between the datapoint where the feedback was given $\boldsymbol{X}^i_{\text{test}}$ and $\boldsymbol{x}$ (the datapoint where the model is applied) and ζ is a smoothing function, for example linear (i.e., for averaging $\zeta(x) = x/F$) or sigmoid. Finally, we combine the feedback-based model with an ML model learned from data as a weighted average with weight α:

$$C(\boldsymbol{x}) = \alpha \text{M}(\boldsymbol{x}, \boldsymbol{\theta}_M) + (1 - \alpha) C_{\text{feedback}}(\boldsymbol{x}, \boldsymbol{\theta}_f), \tag{8}$$

where $\boldsymbol{\theta}_M$ is a vector of the parameters of the machine learning algorithm M, and $\boldsymbol{\theta}_f$ is a vector of parameters of the feedback model. $C(\boldsymbol{x})$ serves as the estimate of how problematic a particular situation is. This value is not a calibrated probability, but is useful for ranking the workstations.

4.3 Multiple domains and decision rules

In practice, it is necessary to answer two questions: *(i)* when a new decision rule appears for a particular domain, should it be used in other domains? *(ii)* when a new domain appears (a new set of workstations joined the system), which of the existing decision rules should we use? The first question can be addressed by comparing historical performance with and without this rule for each domain. This operation can be performed efficiently because the model and decision rule outputs on the historical data can be cached and reused; the rest can be parallelized.

A naive algorithm cannot effectively resolve the second problem. It would need to compare all the subsets of the collected decision rules, which will result in $O(2^{|R|}E)$ time complexity, where $|R|$ is the number of unique decision rules, and E is the complexity of evaluating the decision rules on the training dataset and, even if an efficient algorithm existed, choosing from many alternatives without overfitting would require a big data set which may not always be available. Instead, we propose an approach that exploits similarities between domains using maximum mean discrepancy (MMD),

$$\text{MMD}(P, Q) = \left\| \mathbb{E}_{X\sim P}[\varphi(X)] - \mathbb{E}_{Y\sim Q}[\varphi(Y)] \right\|_{\mathcal{H}}, \tag{9}$$

where the distributions P and Q are defined over domain $\mathcal{X}$, and φ is a feature map from $\mathcal{X}$ to Hilbert space $\mathcal{H}$ [16].

Algorithm. First, we sort all domains by the MMD similarity to the novel domain. Then, we iterate through the domains and add decision rules by batches for each domain if they increase predictive performance. Otherwise, we add the domains to queue ρ. Next, we iterate through the queue and add each decision rule individually if it increases the model's performance. The whole procedure is shown in Algorithm 1.

Algorithm 1 Decision rule selection for a novel domain.

Input: new domain d_{new}, set of domains: $\mathcal{D}$, decision rules for each domain R, DRE model M, historical data for each domain H;
The algorithm uses two external functions:
Perf: a function that takes a model, historical data, and the list of predicates. Returns: predictive performance (performs k-fold cross-validation).
MMD: takes two data samples and returns a measure of similarity between the distributions.

▷ Sort domains by MMD with d_{new}
$\mathcal{D} \leftarrow$ sort($\mathcal{D}$, key=λd[MMD($H[d_{\text{new}}], H[d]$)])

▷ Iterate over domains and add rules as a batch
$\rho \leftarrow \emptyset$; ▷ queue of not added domains for the next phase;
for $d \in \mathcal{D}$ **do**
  $p_{\text{new}} \leftarrow \text{perf}(M, H[d_{\text{new}}], R[d_{\text{new}}])$;
  $p_{\text{updated}} \leftarrow \text{perf}(M, H[d_{\text{new}}], R[d_{\text{new}}] \cup R[d])$
  $\delta \leftarrow |p_{\text{new}} - p_{\text{updated}}|$;
  if $\delta > 0$ **then**
    $R[d_{\text{new}}] \leftarrow R[d_{\text{new}}] \cup R[d]$;
  else
    $\rho \leftarrow \rho \cup d$;
  end if
end for

▷ Add the rest of the rules in a greedy manner
for $d \in \rho$ **do**
  for $r \in R[d]$ **do**
    $\delta \leftarrow \ldots$ ▷ compare performance with and without r;
    if $\delta > 0$ **then**
      $R[d_{\text{new}}] \leftarrow R[d_{\text{new}}] \cup \{r\}$;
    end if;
  end for
end for
Return: R;

Workstation ID	Pred. prob. malfunctioning	DR_1	DR_2	DR_3	DR_4
ID_1	0.92	✓	–	–	–
ID_2	0.87	✓	–	✓	–
ID_3	0.74	–	✓	–	–
ID_4	0.72	–	–	–	–

Figure 2: Schematic visualization of the outputs provided to the domain experts. The first column identifies the workstations, ordered by the model's predictions (the second column). The second column shows the (uncalibrated) probability of malfunctioning. Columns DR_i are the results of elicited rules, selected by domain experts as the most interpretable.

Complexity. The proposed algorithm works with time complexity $O(E \max(|R|, |\mathcal{D}|))$.

Assumptions and optimality. For a fixed model M, we will call a decision rule r *aligned* with a domain d_i: $r \sim d_i$, if exclusion of r makes the whole model less accurate on the domain d_i. Algorithm 1 returns a quasi-optimal allocation of decision rules under the following assumptions: for a new domain d_{new} and for processed domains $d_1, \ldots, d_k$, such that

$$\text{MMD}(d_1, d_{\text{new}}) \leq \ldots \leq \text{MMD}(d_i, d_{\text{new}}) \leq \ldots \leq \text{MMD}(d_k, d_{\text{new}}), \quad (10)$$

there exists j, such that domains $d_1, \ldots, d_j$ are close to d_{new} ($\forall r \in R[d_{1 \leq i \leq j}] : p(r \sim d_{\text{new}}) = 1$), and $\forall r \in R[d_{j<i\leq k}] : p(r \sim d_{\text{new}}) = \epsilon$ for a small ϵ (the probabilities are calculated over all model specifications, including different combinations of decision rules from R). We can see that by the design of the algorithm we first add the rules, only if they are aligned with the novel domain, starting from the closest domains. By the construction of this algorithm, the model's performance does not decrease while adding decision rules from other domains. When the first loop is over, variable ρ stores only the domains where all the rules were not aligned with d_{new}. By the second loop we add those that became aligned in combination with other domains. When we add the rest of the decision rules linearly, we neglect the higher-order of epsilon probabilities of combining several rules to be beneficial for generalization performance. Even though the assumptions are quite strong, we found them to be realistic and helpful in cold starting novel domains.

4.4 Recommender engine for predictive maintenance

We combine the results produced by the ML algorithm and elicited decision rules into a ranked list for domain experts. The system predicts the probability of a device malfunctioning (the second column in Fig. 2) and shows the ranked list of the devices ordered by the probabilities. The outputs are enriched with additional information that helps experts to make faster decisions and navigate the list effectively. We used outputs of the decision rules as the additional information (columns named DR_i in Fig. 2). Domain experts can

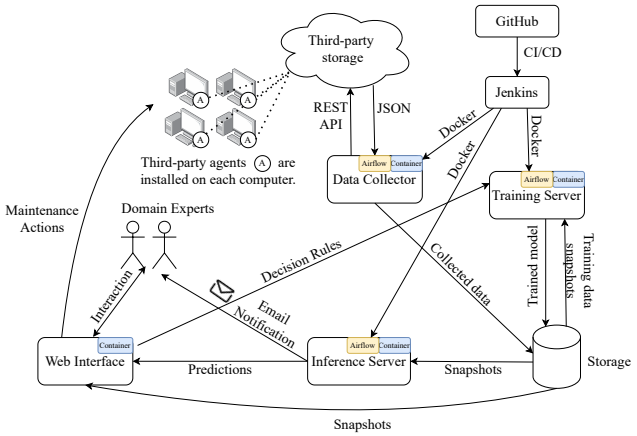


Figure 3: Interaction of the services and agents in the developed system. The system's end goal is to predict future workstation (on the left of the picture) problems and resolve them proactively. The system functioning starts from the source code storage (illustrated in the right upper corner). The code is built by Jenkins and distributed as Docker containers to the data collector, training, and inference services (in the middle of the image). The inference server notifies the domain experts via email notifications daily, and domain experts perform actions based on these predictions. The domain experts return their decisions to the training server and perform maintenance through the web interface.

vary their load by selecting the number of predictions shown. Moreover, they can aggregate similar problems with SQL-like queries to navigate the list faster. An example of such query, to select only workstations with the probability of malfunction higher than 0.9 and with active decision rules 17 and 23, is

```
SELECT * FROM CUR_TICKETS WHERE
                P_M > 0.9 AND DR_17 AND DR_23 .
```

5 INDUSTRIAL IMPLEMENTATION

In practice, it is not enough to develop a model with decent predictive performance; it is necessary to build the whole system with continuous data delivery, fault tolerance, and effortless deployment of new versions to the end-users. Here, we describe the design of our system. The system consists of three principal components: a data collector, training, and inference services, as shown in Fig. 3. The process can be described as follows: Third-party software collects measurements from the workstations and stores them in the third-party cloud. Our data collector extracts the data from the REST API and puts it to S3-compatible storage that is implemented via Ceph[1]. The data is stored in a raw format, roughly 1.2 Tb during a year. The second component (the training service) extracts and preprocesses data from S3, trains a machine learning model, and saves the serialized model to S3 storage. The inference service delivers predictions by running the machine learning model on the most recent data and returning the predictions to domain experts. Our system also provides the predictions to domain experts daily, and the training service runs as an Airflow job. The services are

[1]https://ceph.io/

deployed as docker containers[2] that are orchestrated via Kubernetes [3]. Docker containers are built by Jenkins server [4] where the new versions of code are delivered by GitHub continuous delivery. The model versioning is done via versioning the inference service codebase and changing the newly trained model's path. The same approach can be extended to creating a real-time service by horizontal scaling. Also, the analysis of the workstations is delivered via a web interface, where domain experts can give decision rule feedback, use a tool for visual exploration, or connect to the broken workstation.

5.1 Interface for navigation

An additional feature of the developed system is the possibility to visually navigate over the space of the workstations. The navigation process helps systematically identify problematic workstations and fix them with long-term measures. Fig. 4 shows the mockup version of the interface for domain experts and t-distributed stochastic neighbor embedding (t-SNE) for three different companies. For the second company, we also show workstations identified as chronically problematic by domain experts with red circles. Having this information, domain experts can zoom in on the visualizations in the areas close to systematically problematic workstations and explore similar workstations.

5.2 Monitoring

The deployed model has to be monitored. In our case, the predictions are provided to the domain experts daily, so one applicable metric is the number of valuable tickets (actionable or marked by domain experts as useful) per time interval. When this metric starts to fluctuate, it is a strong signal of a problem with the model, and either the model should be re-trained, the parameters should be adjusted, or the experts should be notified.

6 EXPERIMENTS

We performed offline and online (with real customers) validation of the deployed service. For offline validation, we measured the model's performance on simulated and historical data (collected from customers' workstations). During online validation, we surveyed domain experts about real-life problems predicted by the proposed model.

6.1 Offline experiments

6.1.1 Data simulator. We developed a synthetic data simulator based on survival analysis to generate an artificial dataset. The simulator facilitates reproducibility and provides a controllable environment for PdM experiments.

In the simulator, we assume that for each piece of equipment, a survival function $s(\lambda_i, t)$ exists. The survival function is a function that shows the probability of flawless performance ("survival") of the i-th workstation up to time t. We use $s(\lambda_i, t) = exp(-\lambda_i t)$, where λ_i is a constant hazard function specific for each workstation. We also introduce a parameter ρ_i that shows the delay between experiencing problems and recovery. We assume that there are N workstations, D features describing each of them, and T timestamps (in our experiments, $N = 1000$, $D = 12$, and $T = 100$). We model both categorical and continuous features.

Continuous features. We assume that each continuous feature has two modes of functioning: normal and abnormal. We define features $x_i^{d,t}$ for the i-th workstation, d-th feature, and t-th timepoint as

$$x_i^{d,t} = \gamma(3, \lambda_i, t)\mathcal{N}_+ + (1 - \gamma(3, \lambda_i, t))\mathcal{N}_-,$$
$$\gamma(a, \lambda, t) = \frac{a^{s(\lambda, t-t_r)} - 1}{a - 1}, \mathcal{N}_+ = \mathcal{N}(\mu_+, \sigma_+), \mathcal{N}_- = \mathcal{N}(\mu_-, \sigma_-), \tag{11}$$

where t_r is the largest recovery time $\leq t$, and γ is a balancing coefficient between normal and abnormal behaviour. Parameters μ and σ are sampled from some distributions (these distributions are parameters of the simulator). This approach is motivated by exploring the behavior of the features in the real system: informative features tend to change their absolute values towards breaks.

Categorical features. Categorical features also contained two modes: normal and abnormal. Close to the point of malfunction, categorical variables switch from normal to abnormal mode. The speed of the switch is regulated by λ_i and ρ_i parameters and a stochastic component.

In total, we modeled the distribution with eight categorical features and four continuous features. The generated data for one workstation is visualized in Fig. 5.

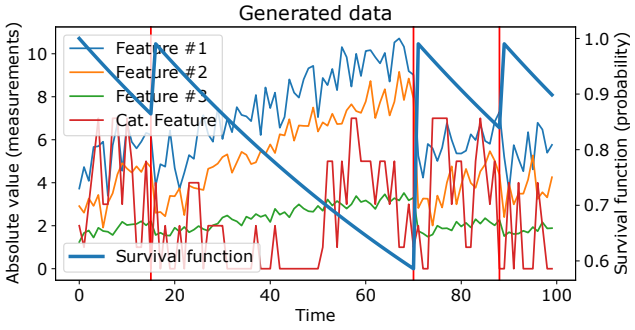
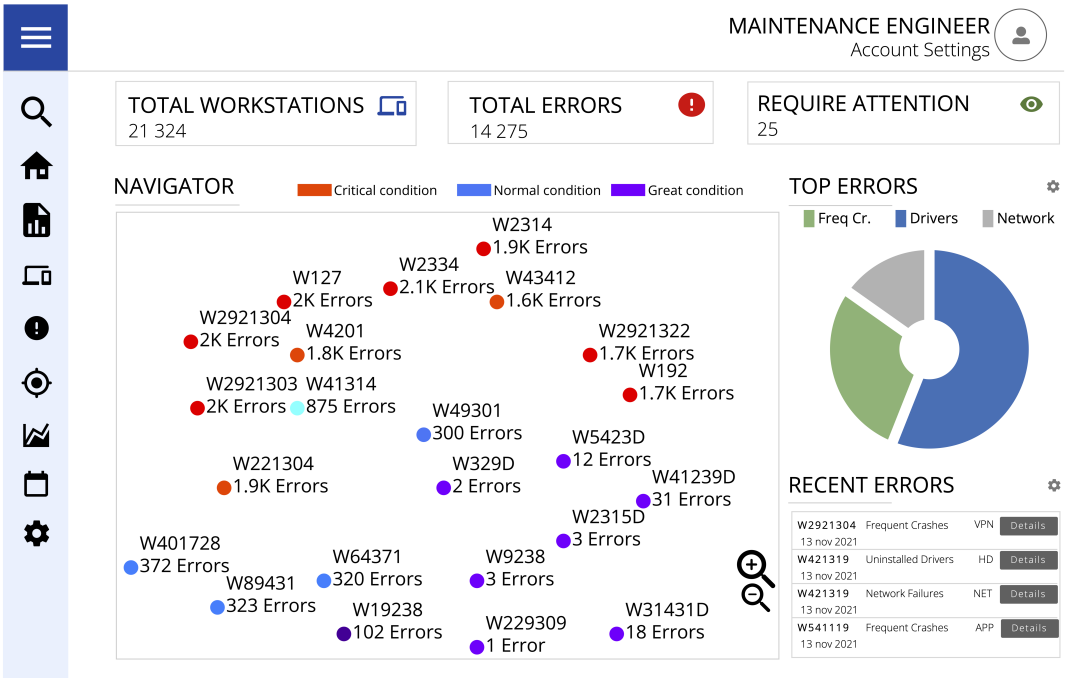


Figure 5: Generated data for one workstation. It is evident that three generated features increase toward the point of repair (marked with red vertical lines) and, then, restore their values.

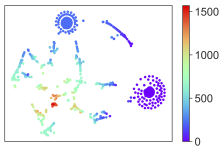
We performed an experimental comparison of the methods described earlier in this paper. We first compared the ML methods and observed the following results: logistic regression (f_1-score = 0.2581 ± 0.0121), extremely randomised trees (f_1-score = 0.2796 ± 0.0234), and gradient boosting (f_1-score = 0.3425 ± 0.0229). By regulating signal to noise ratio via the simulator's parameters the results can be changed. We observe that the ranking of the models evaluated on synthetic data is similar to evaluated on historical data in Sec. 6.1.2, which shows the practical utility of the simulator.

We also modeled artificial domain experts via decision trees with low depth. Each human was parameterized with experience (the fraction of seen data from the training dataset) and depth (the complexity of decision rules produced by the expert), and included them as it is described in Sec. 4.2. We used depth $\sim \mathcal{U}(2, 8)$

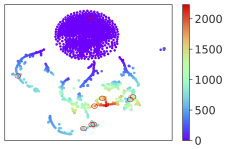
[2]https://www.docker.com/
[3]https://kubernetes.io/
[4]https://www.jenkins.io/



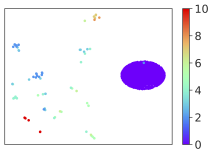
(a) The navigation interface (mockup version). Users zoom in the visualized space of the alerts and explore areas with a high density of problematic workstations.



(b) Company #1.



(c) Company #2.



(d) Company #3.

Figure 4: The interface for browsing the alerts in the visualized space of workstations. Image (a) shows the interface where workstations are the points. Each workstation is marked with colors that depend on how many alerts were collected for the workstation (in all figures). Navigating this space allows domain experts to identify similar workstations and fix them. The interface also shows the most frequent problems in the set of workstations as a pie chart on the right. Furthermore, in the right bottom corner, it shows the latest errors with a possibility to further investigate the issues. The visualization in (a) is a zoomed-in version of all workstations, with full plots of three companies shown in (b)-(c). The plots have been made with t-SNE [30] which places similar data points close to each other and reveals clusters. Based on the coloring, clusters with problematic workstations can be easily identified as the red clusters in the images.

and experience $\sim \mathcal{N}(0.75, 0.05)$. Decision rule elicitation improved f_1-score to 0.5654 ± 0.0202.

Our experiments indicated that with the growth in the number of feedback rules or the growth in the number of modeled domain experts, the predictive performance improves (see Appendix).

6.1.2 Historical data. We collected data from 1075 employee workstations at an undisclosed company for 21 days in total. The data were collected bi-hourly and contained the workstation measurements required for ML modeling and information about alerts, breaks, and user complaints.

We applied Bayesian optimization (as described in Sec. 3.2) with the UCB acquisition function, the RBF kernel for the Gaussian process, $\kappa = 4$, and the decay coefficient $\kappa_{decay} = 0.99$. We used 30 initialization points and ran the optimization process for 20 iterations (however, we observed the convergence earlier). As the target function, we used the mean of f_1-score estimated by 10 evaluations with different train/test splits. The gradient boosting approach showed the best results across all the models; we found it to be consistently better than other models across the different values

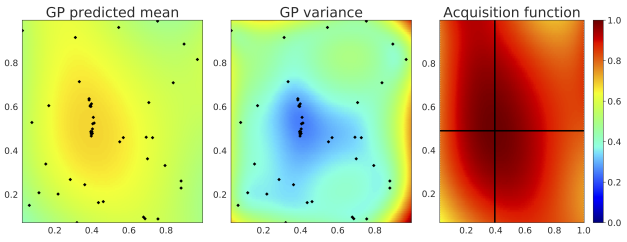


Figure 6: Bayesian optimization of the information and effect periods. Here, the axes are normalized information and effect periods, and the target function is the f_1-score. The colors show the GP mean on the left, the variance in the middle, and the acquisition function on the right. The acquisition function plot shows the next selected point during BO.

of T_{inf}, T_{eff}, and the threshold values; thus, we used it to estimate the quality of the proposed values of T_{inf} and T_{eff}. As a result of BO, we obtained optimal values for the information and effect periods:

Table 1: Offline Experiments (10% threshold).

Algorithm	Inf. period	Eff. period	f_1-score	precision	recall
Logistic regression	48h	48h	0.53 ± 0.01	**0.87 ± 0.02**	0.38 ± 0.01
ExtraTrees classifier	48h	48h	0.65 ± 0.03	0.58 ± 0.03	**0.73 ± 0.02**
Gradient boosting	48h	48h	0.75 ± 0.01	0.84 ± 0.02	0.68 ± 0.02
DRE (3)	48h	48h	0.75 ± 0.01	0.86 ± 0.02	0.67 ± 0.02
DRE (5)	48h	48h	0.76 ± 0.01	0.84 ± 0.02	0.70 ± 0.02
DRE (20)	**48h**	**48h**	**0.78 ± 0.01**	0.85 ± 0.02	**0.73 ± 0.02**
Logistic regression	31.9h	119.9h	0.51 ± 0.02	0.88 ± 0.02	0.35 ± 0.01
ExtraTrees classifier	31.9h	119.9h	0.76 ± 0.01	0.72 ± 0.02	**0.82 ± 0.01**
Gradient boosting	31.9h	119.9h	0.77 ± 0.01	0.87 ± 0.02	0.69 ± 0.02
DRE (3)	31.9h	119.9h	0.77 ± 0.01	**0.89 ± 0.02**	0.67 ± 0.02
DRE (5)	31.9h	119.9h	0.78 ± 0.01	0.87 ± 0.02	0.71 ± 0.02
DRE (15)	31.9h	119.9h	0.80 ± 0.01	0.87 ± 0.02	0.74 ± 0.02
DRE (20)	**31.9h**	**119.9h**	**0.81 ± 0.01**	0.84 ± 0.02	0.77 ± 0.02

$T_{\text{inf}} = 31.9h$ and $T_{\text{eff}} = 119.9h$. One of the Bayesian optimization iterations is visualized in Fig. 6.

Next, we evaluated the performance of the machine learning approaches described in Sec. 4.1. We measured the f_1-score, the precision and recall of the algorithms and relied on the f_1-score in the model selection process. We split 30% of the dataset as a hold-out test set and performed test evaluation ten times with different random test sets and random seeds to obtain the confidence intervals. We also checked the results for different thresholds by the total number of alerts: 1%, 5%, and 10%. We found that the models' f_1-score was more than 0.7 for the 5% and 10% thresholds, with a drop to 0.52 for 1%.

The experimental results are presented in Table 1. DRE(n) is the proposed decision rule elicitation method applied to the gradient boosting model over decision trees with n feedback rules. The experiment shows that decision rule elicitation improves the performance (f_1-score), and the improvement grows with the number of the feedback rules. The results are consistent between randomly selected information and effect periods and optimal values obtained via Bayesian optimization. The results (not absolute values but the ordering of the models) are also consistent with the simulator experiments of Sec. 6.1.1.

6.2 Online experiments

To validate the system in a production environment, we trained the model on one-month data. Then, our algorithm generated the predictions for domain experts for one month. Our analysis demonstrated that the predictive performance of the algorithm was consistent with offline testing — the predictive performance ranking of the selected algorithms remained the same. After this, we selected several days and checked the predictions with customers to verify that the problems affected customers' experience during T_{eff}. We evaluated these predictions with domain experts and customers for nine days. The number of checked tickets and days per week was not controlled.

The results of the online validation are presented in Fig. 7. The validation shows the practical usefulness of the proposed approach: the average number of user requests per day measured for the last

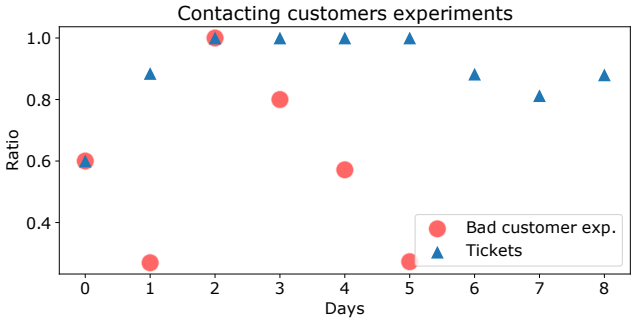


Figure 7: Online experiments. For randomly selected nine days over three weeks, we performed: *(i)* analysis on how many predictions resulted in a ticket creation (domain experts were able to identify a problem, ▲) and *(ii)* how many predictions matched negative user experience (●). We can see that the majority of explored predictions resulted in a ticket; also, significant proportions of those were connected to the negative customer experience. The experiments were conducted for eight days (a longer period could cause a major overload on domain experts and affect customers). The total number of investigated tickets varied from three to 26 tickets per day.

2.5 years for this company was approximately six, which, compared to the results (with more than three tickets per day), promises a long-term reduction in the number of tickets of up to 50%. Currently, the proposed system has been deployed for a year without technical issues and used by domain experts, which shows the validity of system design and modeling choices.

The validation took approximately four human hours per week, which included contacting the customers. We argue that the approach scales well with the growth in the number of workstations because the number of domain experts needed to support operations scales linearly with the number of devices (we assume that network maintenance is a separate task).

7 CONCLUSION

This work presents a scalable machine learning approach to predictive workstation maintenance. The key feature of our approach is close interaction between human experts and machine learning algorithms. The synergy between expert knowledge and artificial intelligence leads to a scalable and robust method for modeling future failures in computer equipment. All the steps were designed to effortlessly generalize the approach to other domains or more equipment, leading to accessible horizontal and vertical scaling. We demonstrated the implementation of our approach to a production case and reported experimental results. Even though we considered only workstation PdM problems, our system can be applied to other PdM tasks without drastic changes.

Our HITL approach was successful both with historical data and in practice. The combination of ML models and decision rule feedback improved the results in the experimental setup and allowed experts to influence the model results. During the online validation, we detected and fixed the problems that affected the users; the observed performance is sufficient to reduce the total number of user complaints by up to 50 percent in the long term. The notification system has been working for one year, which proves the effectiveness of chosen design and implementation.

Our system is a significant advance towards automated, reliable problem management, which could impact a large variety of business sectors.

ACKNOWLEDGMENTS

This work was supported by the Academy of Finland (Flagship programme: Finnish Center for Artificial Intelligence FCAI) and UKRI Turing AI World-Leading Researcher Fellowship, EP/W002973/1. We also acknowledge the computational resources provided by the Aalto Science-IT Project from Computer Science IT.

REFERENCES

[1] Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. 2014. Power to the people: The role of humans in interactive machine learning. *Ai Magazine* 35, 4 (2014), 105–120.

[2] Joeran Beel, Stefan Langer, Marcel Genzmehr, Bela Gipp, Corinna Breitinger, and Andreas Nürnberger. 2013. Research paper recommender system evaluation: a quantitative literature survey. In *Proceedings of the International Workshop on Reproducibility and Replication in Recommender Systems Evaluation*. Association for Computing Machinery, New York, NY, USA, 15–22.

[3] Eric Brochu, Vlad M Cora, and Nando De Freitas. 2010. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599* abs/1012.2599 (2010).

[4] R. Casado-Vara, P. Novais, A. B. Gil, J. Prieto, and J. M. Corchado. 2019. Distributed Continuous-Time Fault Estimation Control for Multiple Devices in IoT Networks. *IEEE Access* 7 (2019), 11972–11984.

[5] Irene Celino. 2020. Who Is This Explanation for? Human Intelligence and Knowledge Graphs for eXplainable AI.

[6] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Association for Computational Linguistics, Doha, Qatar, 103–111. https://doi.org/10.3115/v1/W14-4012

[7] Mihyun Chung and Jaehyoun Kim. 2016. The internet information and technology research directions based on the fourth industrial revolution. *KSII Transactions on Internet and Information Systems (TIIS)* 10, 3 (2016), 1311–1320.

[8] Federico Civerchia, Stefano Bocchino, Claudio Salvadori, Enrico Rossi, Luca Maggiani, and Matteo Petracca. 2017. Industrial Internet of Things monitoring solution for advanced predictive maintenance applications. *Journal of Industrial Information Integration* 7 (2017), 4–12.

[9] Katalin Csilléry, Michael GB Blum, Oscar E Gaggiotti, and Olivier François. 2010. Approximate Bayesian computation (ABC) in practice. *Trends in Ecology & Evolution* 25, 7 (2010), 410–418.

[10] Santanu Das. 2013. Maintenance action recommendation using collaborative filtering. *International Journal of Health Policy and Management* 4, 2 (2013), 7–12.

[11] David Duvenaud. 2014. *Automatic model construction with Gaussian processes*. Ph. D. Dissertation. University of Cambridge.

[12] Eduard Fosch-Villaronga, Pranav Khanna, Hadassah Drukarch, and Bart HM Custers. 2021. A human in the loop in surgery automation. *Nature Machine Intelligence* 3 (2021), 1–2.

[13] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of Statistics* 29 (2001), 1189–1232.

[14] Pierre Geurts, Damien Ernst, and Louis Wehenkel. 2006. Extremely randomized trees. *Machine Learning* 63, 1 (2006), 3–42.

[15] Mary A Gravette and Kash Barker. 2015. Achieved availability importance measure for enhancing reliability-centered maintenance decisions. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability* 229, 1 (2015), 62–72.

[16] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. 2012. A kernel two-sample test. *The Journal of Machine Learning Research* 13, 1 (2012), 723–773.

[17] C. Gu, Y. He, X. Han, and Z. Chen. 2017. Product quality oriented predictive maintenance strategy for manufacturing systems. In *2017 Prognostics and System Health Management Conference (PHM-Harbin)*. IEEE, 1–7.

[18] Liang Guo, Naipeng Li, Feng Jia, Yaguo Lei, and Jing Lin. 2017. A recurrent neural network based health indicator for remaining useful life prediction of bearings. *Neurocomputing* 240 (2017), 98–109.

[19] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[20] Andreas Holzinger. 2016. Interactive machine learning for health informatics: when do we need the human-in-the-loop? *Brain Informatics* 3, 2 (2016), 119–131.

[21] Karen Sparck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation* 28, 1 (1972), 11–21.

[22] Wenchao Li, Dorsa Sadigh, S Shankar Sastry, and Sanjit A Seshia. 2014. Synthesis for human-in-the-loop control systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 470–484.

[23] Jonas Močkus. 1975. On Bayesian methods for seeking the extremum. In *Optimization techniques IFIP technical conference*. Springer Berlin Heidelberg, Berlin, Heidelberg, 400–404.

[24] Robert Munro Monarch. 2021. *Human-in-the-Loop Machine Learning: Active learning and annotation for human-centered AI*. Simon and Schuster.

[25] Alexander Nikitin and Samuel Kaski. 2021. Decision Rule Elicitation for Domain Adaptation. In *26th International Conference on Intelligent User Interfaces*. Association for Computing Machinery, New York, NY, USA, 244–248.

[26] Rahul Potharaju, Navendu Jain, and Cristina Nita-Rotaru. 2013. Juggling the jigsaw: Towards automated problem inference from network trouble tickets. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. USENIX Association, Lombard, IL, 127–141.

[27] Yongyi Ran, Xiaoxia Zhou, Pengfeng Lin, Yonggang Wen, and R. Deng. 2019. A Survey of Predictive Maintenance: Systems, Purposes and Approaches. *ArXiv* abs/1912.07383 (2019).

[28] Lei Ren, Yaqiang Sun, Jin Cui, and Lin Zhang. 2018. Bearing remaining useful life prediction based on deep autoencoder and deep neural networks. *Journal of Manufacturing Systems* 48 (2018), 71–77.

[29] Abhinav Saxena and Kai Goebel. 2008. Turbofan engine degradation simulation data set. *NASA Ames Prognostics Data Repository* (2008), 1551–3203.

[30] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).

[31] J. Wang, C. Li, S. Han, S. Sarkar, and X. Zhou. 2017. Predictive maintenance based on event-log analysis: A case study. *IBM Journal of Research and Development* 61, 1 (2017), 11:121–11:132.

[32] Andrew G Wilson, Christoph Dann, Chris Lucas, and Eric P Xing. 2015. The Human Kernel. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28. Curran Associates, Inc.

[33] Boyuan Yang, Ruonan Liu, and Enrico Zio. 2019. Remaining useful life prediction based on a double-convolutional neural network architecture. *IEEE Transactions on Industrial Electronics* 66, 12 (2019), 9521–9530.

[34] Kazuyoshi Yoshii, Masataka Goto, Kazunori Komatani, Tetsuya Ogata, and Hiroshi G Okuno. 2008. An efficient hybrid music recommender system using an incrementally trainable probabilistic generative model. *IEEE Transactions on Audio, Speech, and Language Processing* 16, 2 (2008), 435–447.

[35] Wubai Zhou, Wei Xue, Ramesh Baral, Qing Wang, Chunqiu Zeng, Tao Li, Jian Xu, Zheng Liu, Larisa Shwartz, and Genady Ya. Grabarnik. 2017. Star: A system for ticket analysis and resolution. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, New York, NY, USA, 2181–2190.